MIDDLEWARE AND CONTROL OF CYBER-PHYSICAL SYSTEMS: TEMPORAL GUARANTEES AND HYBRID SYSTEM ANALYSIS

BY

KYOUNG-DAE KIM

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Doctoral Committee:

Professor P. R. Kumar, Chair
Professor Tamer Başar
Associate Professor Tarek F. Abdelzaher
Assistant Professor Sayan Mitra

ABSTRACT

A cyber-physical system is a system which encompasses computing, communication, and physical entities with emphasis on their interactions. In this work, we study two different aspects of cyber-physical systems: mechanisms as well as policy. At the mechanism end we study how to provide a platform for developing cyber-physical systems, and at the policy end we study how to analyze the performance of the overall control system.

For the purpose of verification of certain high-level properties, a cyber-physical system can be viewed as a hybrid system. It is indeed useful to do so since properties such as safety can then be studied through a reachable set computation. Towards this end, we propose a theory for an over-approximation of the reachable set for a special class of hybrid systems, over a finite time interval, under a certain deterministic and transversal discrete transition condition. A prototype software tool based on the proposed theory is also implemented and used to demonstrate the computation of such an over-approximate reachable set.

At the mechanism end, we address the issue of the design and implementation of real-time mechanisms in Etherware, a middleware developed in the Information Technology Convergence Laboratory at the University of Illinois, so that it can be more suitable for networked control applications. Cyber-physical systems are often safety critical, and so it is important that their performance be predictable. Of particular interest is temporal predictability of interactions, which it is the goal of the middleware to ensure. Also of importance for developing complex control systems is that the middleware be flexible. We address the problem of enhancing these two attributes, and demonstrate the flexibility and temporal predictability of the enhanced Etherware through a networked inverted pendulum application.

To my family, for their love and support

ACKNOWLEDGMENTS

I was able to finish this dissertation thanks to their contributions of many people. In particular, I would like to deliver my sincere appreciation to Professor P. R. Kumar for his invaluable advice on every aspect of my research. I also would like to thank Professors Kang G. Shin, Marco Caccamo, Christoper D. Gill, Lui Sha, and Paulo Tabuada for discussions on the design of real-time scheduling mechanisms for Etherware. For the experiment with an inverted pendulum system, I am very grateful to Daniel J. Block for his generous support in building the system. It was also my true pleasure to have an opportunity to collaborate with Professor Sayan Mitra on the hybrid system verification research. This dissertation has indeed benefited from his expertise on the subject. Last but not least, there is no doubt that I would have never been able to accomplish this dissertation without the unconditional support of my lovely wife Sunghee and my precious son Byungjin.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A cyber-physical system is a system which exhibits a tight integration and interaction between computing and physical entities over a communication network. Depending on the issue of interest at hand, a cyber-physical system can be examined from different viewpoints. When it is the distributed sensing and action of physical entities over the network that is under examination, it can be regarded as a networked control system. On the other hand, if we are interested in understanding the dynamic behavior resulting from the interaction between discrete and continuous dynamics, it can be regarded as a hybrid system. Another dichotomy is that between mechanisms and policies. By "mechanism," we mean *how* the system is to be implemented, and by "policy" we mean *what* is implemented. Again, both issues are important.

In this work, we study cyber-physical systems from all of these viewpoints. We begin by addressing the issue of safety of cyber-physical systems. This can be modeled as one of determining the reachable set and examining whether that set intersects an undesirable subset of the state space. Typically the overall system consists of both continuous as well as discrete interactions. We therefore address the problem of reachable set computation for a special class of hybrid systems which is of interest in several applications. Next, we address the problem of design and implementation of a software platform on which a networked control system can be implemented. We believe that such a software platform with appropriate architecture and mechanisms is the key to the proliferation of networked control systems.

1.1 Safety and Reachability of Hybrid Systems

Cyber-physical systems are often safety critical, since the underlying control systems often are. Therefore it is important to assure that the physical

system will be in a safe state under any functionally correct behavior of the computing system. This of course is also dependent on the reliability of the computing system, which can generally be improved through the use of several different methodologies such as formal verification, fault tolerant design of hardware and software systems, experimental validation, and so on. Assuming therefore that the underlying computing system is functionally correct, a hybrid system can be regarded as a high level abstraction of a cyber-physical system.

Such a hybrid system basically consists of a discrete state transition system, which represents the dynamical behavior of computational entities, and differential equations, which describe the continuous dynamics of physical entities. Its safety verification can typically be addressed through a reachable set computation. However, computing the reachable set of a hybrid system is a non-trivial problem in general due to the complex interaction between dynamics in two different domains. In fact, even for a hybrid system whose continuous dynamics is linear, deterministic, and time-invariant, it is not known that computation of its reachable set is a decidable problem.

In this thesis, we study this problem, making progress by identifying a framework that is both tractable and constitutes a model class that is useful for several applications. More precisely, we show that it is possible to compute an over-approximation of a reachable set for a class of hybrid automata, which we call *Deterministic Transversal Linear Hybrid Automaton (DTLHA)*, for a finite time, with arbitrarily small approximation error, even though computing the exact reachable set for such a hybrid automata is still not computationally feasible. Moreover, we also show that such an over-approximation can still be computed even without requiring infinite precision calculations of the underlying numerical operations. However, in this case, the smallest approximation error that can be achieved is limited by the accuracies that the underlying numerical operations support. An architecture is also proposed for implementing a software tool to compute a reachable set of a DTLHA. It is designed to decouple the basic algorithm, which is also proposed based on our theoretical framework, from the choice of several runtime adaptations that are needed by the basic algorithm to continue its computation. An example of DTLHA is considered to illustrate the capability of a prototype implementation of the proposed algorithm.

1.2 A Real-Time Middleware for Networked Control Systems

In general, a networked control system is a system whose constituents such as sensors, actuators, and controllers are distributed over a communication network, and their corresponding control loops are formed through a network layer. Thus, the scale of the networked control system is typically much larger than that of classical control systems. In addition to the scale of the system, the complexity of a networked control system is also greater. Due to the existence of the networked communication and computation system below the control application layer, several challenging issues such as communication delays, the interface between a control application and the network layer, platform heterogeneity, clock differences between the computers, and others, arise. Clearly, all of these constitute an extraordinarily big burden on control engineers if they have to address these issues too while designing the control loops. One solution to release these burdens from the control engineer is to interpose an abstraction layer between the application layer and the underlying networked communication and computation layer. Such an abstraction layer can encapsulate the complexity of the underlying system so that the application layer can have a much simpler view of the system. This can significantly simplify and shorten the development of a networked control application. Typically, such an abstraction can be realized as a software framework, called a *middleware*.

An early version of such a middleware, called *Etherware* [1], has been developed and implemented in the Information Technology Convergence laboratory at the University of Illinois. Etherware has an architecture and mechanisms so that it provides most of the functionalities required for distributed applications. However, the Etherware of [1] still needed to be enhanced in some aspects to be used for networked control systems; for example, it does not provide any temporal guarantees at all. Since any delay and jitter in sensing and control action in the control loop can affect the performance or even the stability of a physical system, providing temporal guarantees is an essential functionality which needs to be supported by any software framework used for control systems.

To address this issue, we propose and implement a new scheduling mechanism of Etherware to improve its performance by making feasible the de-

velopment of control applications that require temporal predictability. We demonstrate that the enhanced Etherware does indeed provide both flexibility and temporal predictability, and validate it through experiments on a networked inverted pendulum system. We demonstrate, for the first time, we believe, the facility of migration, over a network, of the controller for the pendulum, while preserving stability.

1.3 Organization of the Dissertation

This thesis is organized roughly into two parts. The first part of this thesis consists of Chapters 2 to 5. In this part, the problem of reachability of a special class of hybrid systems is addressed. In the second part of this thesis, which is from Chapters 6 to 8, the issue of design and implementation of a software platform for cyber-physical systems is considered.

In Chapter 2, we first define a special class of hybrid automaton, called Deterministic Transversal Linear Hybrid Automaton (DTLHA) which is a Linear Hybrid Automaton (LHA) satisfying certain types of discrete transition conditions, and a special class of reachable set, called a bounded ϵ-reach set. We show that, for any $\epsilon \in \mathbb{R}^+$, a bounded ϵ-reach set of a DTLHA can be computed under some exact computation assumptions. In Chapter 3, we extend the theory to relax the exact computation assumptions and show that a bounded ϵ-reach set of a DTLHA can still be computed with finite precision calculations. In Chapter 4, we propose an algorithm for a bounded ϵ-reach set computation of a DTLHA based on the theoretical results in previous chapters. We also propose an architecture for a software implementation of the proposed algorithm. In Chapter 5, we first introduce some techniques that have been developed during the implementation of the algorithm to improve the overall capability of the implementation. Then we evaluate the implemented software through an example of DTLHA by computing a bounded ϵ-reach set in several different cases.

Next, in Chapter 6, we first investigate the characteristics of networked control systems and then describe some requirements for a middleware framework for networked control systems. Then we introduce the existing Etherware developed in [1]. The design and implementation of Etherware mechanisms to support the timeliness requirement is discussed in Chapter 7. The

implementation of a networked inverted pendulum system and the evaluation of the performance of the enhanced Etherware on this system are described in Chapter 8.

CHAPTER 2

BOUNDED ϵ-REACHABILITY OF A DETERMINISTIC TRANSVERSAL LINEAR HYBRID AUTOMATON

A hybrid automaton is a convenient mathematical model for systems which undergo both discrete and continuous evolution. Computing the set of reachable states of a hybrid automaton is useful for safety verification and automatic controller synthesis. However, it is well known that computing the exact reach set of a hybrid automaton is undecidable except for hybrid automata with relatively simple continuous dynamics such as timed-automata whose continuous states evolve at a constant rate, multi-rate automata which allow multiple constant rates in their continuous state evolution, and initialized rectangular hybrid automata which have rectangular regions for the rate of the continuous state evolution, which also reset the continuous variable whenever the dynamics corresponding to the variable is changed [2].

To avoid these theoretical limitations, research in hybrid system verification in the recent years has since focused on algorithms computing over-approximations of the reachable states of various classes of hybrid automata [3–14]. As an example, HyTech [15] computes the reach set of rectangular hybrid automata that are defined by linear differential inequalities of the form $A\dot{x} \sim b$, where A is a constant matrix, b is either a constant vector or a constant rectangular region, and $\sim \in \{\geq, \leq\}$. For hybrid automata whose continuous dynamics are more general than those of rectangular hybrid automata, HyTech computes a reach set of the automaton by translating the original model into a rectangular hybrid automaton if the model is translatable. Otherwise, an over-approximated reach set is computed using an approximate automaton that is obtained by relaxing the continuous dynamics of the original automata. In PHAVer [16], some implementation related issues such as excessive complexity, slow convergence, and insufficient accuracy of over-approximations which HyTech suffers from are resolved, so as to handle a larger class of systems such as linear hybrid automata that have affine dynamics. To compute an approximate reach set of linear hybrid

automata, PHAVer performs on-the-fly over-approximation of the reach set, but it does not provide a bound on the degree of the over-approximation. Reference [17] solves a verification problem of a class of hybrid automata called a polyhedral-invariant hybrid automata (PIHA) whose invariants and guards are defined by linear inequalities. PIHA are equivalent to switched continuous dynamics systems in which there are no jumps in the continuous state trajectory. To verify the properties of PIHA from initial states, a finite state transition system, which is a conservative approximation of the original hybrid system, is constructed. If the verification is inconclusive, the constructed finite states are refined and then a new finite state transition system is constructed. This procedure is repeated until the given verification problem is conclusively resolved. The algorithm is not guaranteed to terminate in general. In constructing a finite state transition system, determining a polyhedral approximation of each sampled segment of the continuous state evolution between switching planes is the underlying fundamental technique in the algorithm. A similar idea of polyhedral approximation of continuous state evolution of a continuous linear dynamics system is developed in [18]. In addition to the polyhedral approximation, other useful techniques using ellipsoids and zonotopes for approximating a continuous state evolution are introduced in [19] and [20], respectively.

In this chapter, we address the problem of approximate reach set computation of a class of hybrid systems with deterministic affine dynamics from a single state. Specifically, we propose an algorithm that computes a *bounded ϵ-reach set* from a given initial state, for any arbitrarily small ϵ, up to either an upper bound on time, or an upper bound on the number of discrete transitions, under the assumption of *deterministic* and *transversal* discrete transition. Our approach for constructing an over-approximate continuous state evolution is related to the sampling-based techniques presented in [17] and [18]. However, our approach for approximation is unique in the sense that an over-approximating polyhedron is constructed using only sampled states and sampling period, while others rely on the continuous trajectories between the samples. As in [17], the execution of a hybrid automaton that is defined in Section 2.1 also exhibits a continuous state evolution without having jumps at the times of discrete transition. Thus the hybrid automata models are similar to each other. However, the fundamental distinguishing feature of our approach for solving verification problems is that an approx-

imate reach set is directly computed using a given hybrid automata model, while in [17] a finite state transition system is constructed which has a simulation relation to the original hybrid automaton.

2.1 Deterministic and Transversal Linear Hybrid Automaton (DTLHA)

A Hybrid Automaton (HA) is a non-deterministic state transition system whose (a) discrete state changes instantaneously through discrete transitions, and (b) continuous state evolves continuously over an interval of time. In this chapter, we consider a special class of HAs for which the continuous state space is partitioned into *polyhedral sets*, and the evolution over the continuous state space is *deterministic* and specified by *linear differential equations*. We also assume that when such an HA changes its discrete state, the discrete transition is both deterministic and transversal. We will explicitly define this type of discrete transition later in this section.

More formally, we assume that the continuous state space $\mathcal{X} \subset \mathbb{R}^n$ is closed and bounded, and is partitioned into a collection of polyhedral regions $\mathcal{C} := \{\mathcal{C}_1, \cdots, \mathcal{C}_m\}$, that is

$$\bigcup_{i=1}^{m} \mathcal{C}_i = \mathcal{X}, \quad \mathcal{C}_i^\circ \cap \mathcal{C}_j^\circ = \emptyset \quad for\ i \neq j, \tag{2.1}$$

where m is the size of the partition, each $\mathcal{C}_i$ is a polyhedron, called *cell*, with a nonempty interior, and $\mathcal{C}_i^\circ$ is the interior of $\mathcal{C}_i$. Two cells $\mathcal{C}_i$ and $\mathcal{C}_j$ are said to be *adjacent* if the affine dimension of $\partial\mathcal{C}_i \cap \partial\mathcal{C}_j$ is $(n-1)$, or, equivalently, cells $\mathcal{C}_i$ and $\mathcal{C}_j$ intersect in an $(n-1)$-dimensional facet. Here $\partial\mathcal{C}_i$ denotes the boundary of $\mathcal{C}_i$. Two cells $\mathcal{C}_i$ and $\mathcal{C}_j$ are said to be *connected* if there exists a sequence of adjacent cells between $\mathcal{C}_i$ and $\mathcal{C}_j$.

Definition 1. *An n-dimensional* Linear Hybrid Automaton (LHA), [1] *is a*

[1] In the hybrid system literature [15, 21] the word "linear automaton" has been used to denote a system where the differential equations and inequalities involved have constant right hand sides. However, this does not conform to the standard notion of linearity where the right hand side is allowed to be a function of state. In particular, it does not even include the standard class of linear time-invariant systems that is of central interest in control systems design and analysis. We use the term "linear" in this latter more mathematically standard way that therefore encompasses a larger class of systems, and more importantly, important classes of switched linear systems that are of much interest.

tuple $(\mathbb{L}, Inv, A, u)$ *satisfying the following properties.*

(a) $\mathbb{L}$ *is a finite set of* locations *or* discrete states. *The state space is* $\mathbb{L} \times \mathbb{R}^n$, *and an element* $(l, x) \in \mathbb{L} \times \mathbb{R}^n$ *is called a* state.

(b) $Inv : \mathbb{L} \to 2^{\mathcal{C}}$ *is a function that maps each location to a set of cells,* [2] *called an* invariant set *of a location, such that*

(i) for each $l \in \mathbb{L}$, *all the cells in* $Inv(l)$ *are connected,*

(ii) for any two locations $l, l' \in \mathbb{L}$, $Inv(l)^\circ \cap Inv(l')^\circ = \emptyset$, *and*

(iii) $\cup_{l \in \mathbb{L}} Inv(l) = \mathcal{X}$.

(c) $A : \mathbb{L} \to \mathbb{R}^{n \times n}$ *is a function that maps each location to an* $n \times n$ *real-valued matrix, and*

(d) $u : \mathbb{L} \to \mathbb{R}^n$ *is a function that maps each location to an* n*-dimensional real-valued vector.*

In the sequel, for each $l_i \in \mathbb{L}$, we use A_i, u_i, Inv_i to denote $A(l_i)$, $u(l_i)$, and $Inv(l_i)$, respectively.

An LHA $\mathcal{A}$ defines a trajectory, an execution, and a discrete transition as follows:

Definition 2. *For a location* $l_i \in \mathbb{L}$, *a* trajectory *of duration* $t \in \mathbb{R}_{\geq 0}$ *for an LHA* $\mathcal{A}$ *with* n *continuous dimensions (or variables) is a continuous map* η *from* $[0, t]$ *to* $\mathbb{R}^n$, *such that*

(a) $\eta(\tau)$ *satisfies the differential equation*

$$\dot{\eta}(\tau) = A_i \eta(\tau) + u_i, \tag{2.2}$$

(b) $\eta(\tau) \in Inv_i$ *for every* $\tau \in [0, t]$.

For such a trajectory η, its *duration* is t, and it is denoted by $\eta.dur$. We use Σ_i to denote the linear time invariant (LTI) system defined at a location l_i as in (2.2).

[2] Actually, to be precise, the invariant of a location is the union of such cells; however, we abuse the terminology slightly for ease of reading.

Definition 3. *An* execution α *of an LHA* $\mathcal{A}$ *from a starting state* $(l_0, x_0) \in \mathbb{L} \times \mathbb{R}^n$ *is defined to be the concatenation of a finite or infinite sequence of trajectories* $\alpha = \eta_0\eta_1\eta_2\ldots$*, such that*

(a) $\eta_0(0) = x_0$*,*

(b) $\eta_k(0) = f\left(\eta_{k-1}(\eta_{k-1}.dur)\right)$ *for* $k \geq 1$*, and*

(c) $\alpha.dur = \sum_k \eta_k.dur$

where η_k *represents a trajectory defined at some location* $l \in \mathbb{L}$*,* $f(\cdot)$ *is a reset function, and* $\alpha.dur$ *denotes the duration of an execution.*

Thus, if we restrict our consideration to the executions with an identity reset map, i.e., $\eta_k(0) = \eta_{k-1}(\eta_{k-1}.dur)$ in (b), then we can represent an execution α of an LHA $\mathcal{A}$ from an initial state $(l_0, x_0) \in \mathbb{L} \times \mathbb{R}^n$ for time $[0, t]$ as a continuous map $x : [0, t] \to \mathbb{R}^n$ such that (a) $t = \alpha.dur$, (b) $x(0) = x_0 \in Inv_0$, (c) $x(\tau_k) = \eta_k(0)$, and (d) $x(\tau) = \eta_{k-1}(\tau - \tau_{k-1})$ for $\tau \in [\tau_{k-1}, \tau_k]$, where $\tau_0 = 0$, and $\tau_k = \sum_{i=0}^{k-1} \eta_i.dur$ for $k \geq 1$. Note that τ_k for $k \geq 1$ represents the time at the k-th discrete transition between locations and the continuous state is not reset during discrete transitions.

Definition 4. *For* $l_i, l_j \in \mathbb{L}$*, a* discrete transition *from* l_i *to* l_j *occurs at a continuous state* $x(\tau')$ *at time* τ'*, whenever* $x(\tau') \in Inv_i \cap Inv_j$ *and* $x(\tau') = \lim_{\tau \nearrow \tau'} x(\tau)$ *where* $x(\tau) \in (Inv_i)^\circ$ *for* $\tau \in (\tau' - \delta, \tau')$ *for some* $\delta > 0$*.*

Definition 5. *A discrete transition is called a* deterministic discrete transition *if there is only one location* $l_j \in \mathbb{L}$ *to which a discrete transition state* $x(\tau_k)$ *can make a discrete transition from* l_i*. Furthermore, for* $\epsilon > 0$*, we call a discrete transition a* transversal discrete transition *if the following condition is satisfied at* $x(\tau_k)$*:*

$$\langle \dot{x}_i(\tau_k), \vec{n}_i \rangle \geq \epsilon \quad \wedge \quad \langle \dot{x}_j(\tau_k), \vec{n}_i \rangle \geq \epsilon, \tag{2.3}$$

where $\vec{n}_i$ *is an outward normal vector of* ∂Inv_i *at* $x(\tau_k)$*, and* $\dot{x}_i(\tau_k) = A_i x(\tau_k) + u_i$*, and* $\dot{x}_j(\tau_k) = A_j x(\tau_k) + u_j$ *are the vector fields at* $x(\tau_k)$ *evaluated with respect to the continuous dynamics of location* l_i *and* l_j*, respectively.*

Figure 2.1 illustrates a case when $x(\tau_k)$ satisfies such a deterministic and transversal discrete transition condition. Note that if $x(\tau_k)$ satisfies a deterministic and transversal discrete transition condition, then $x(\tau_k)$ must make

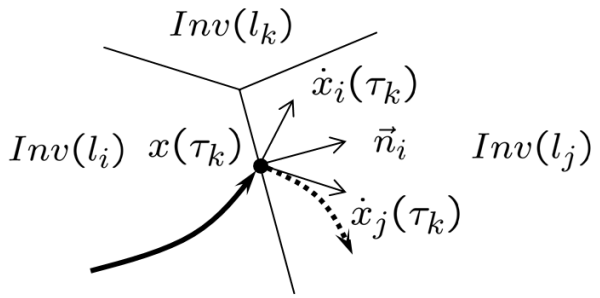


Figure 2.1: A deterministic and transversal discrete transition from a location l_i to a location l_j occurring at $x(\tau_k) \in \partial Inv(l_i) \cap \partial Inv(l_j)$.

a discrete transition from a location l_i to the other location l_j, and l_j has to be unique. Furthermore, the *Zeno behavior*, an infinite number of discrete transitions within a finite amount of time, does not occur if a discrete transition is a transversal discrete transition.

We now define a special class of LHA whose every discrete transition satisfies the deterministic and transversal conditions defined in Definition 5 as follows:

Definition 6. *Given an LHA $\mathcal{A}$, a starting state $(l_0, x_0) \in \mathbb{L} \times \mathbb{R}^n$, a time bound T, and a jump bound N, we call an LHA $\mathcal{A}$ as a* Deterministic Transversal Linear Hybrid Automaton (DTLHA) *if all discrete transitions in the execution starting from x_0 up to time T or up to N transitions (whichever is earlier) are deterministic and transversal.*

2.2 Definition of Bounded ϵ-Reach Set of a DTLHA

In this section, we define the bounded reach set of a DTLHA and its over-approximation, called bounded ϵ-reach set, as follows:

Definition 7. *A continuous state in $\mathcal{X}$ is* reachable *if there exists some time t at which it is reached by some execution x.*

Definition 8. *Given a time t, the* bounded reach set *up to time t, denoted as $\mathcal{R}_t(x_0)$, of a DTLHA $\mathcal{A}$ is defined to be the set of continuous states that are reachable for some time $\tau \in [0, t]$ by some execution x starting from $x_0 \in Inv_0$.*

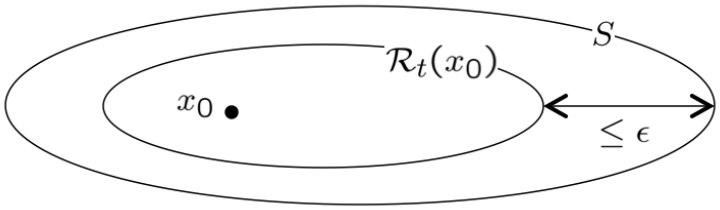


Figure 2.2: The relation between a bounded reach set $\mathcal{R}_t(x_0)$ and its bouned ϵ-reach set S.

Definition 9. *Given $\epsilon > 0$, a set of continuous states S is called a* bounded ϵ-reach set *of a DTLHA $\mathcal{A}$ over a time interval $[0, t]$ from an initial state x_0 if $\mathcal{R}_t(x_0) \subseteq S$ and*

$$d_H(\mathcal{R}_t(x_0), S) \leq \epsilon, \tag{2.4}$$

where $d_H(\mathcal{P}, \mathcal{Q})$ denotes the Hausdorff distance [3] *between two sets $\mathcal{P}$ and $\mathcal{Q}$.*

Figure 2.2 shows the relation between a bounded reach set and its over-approximation for a given approximation parameter $\epsilon > 0$.

The specific norm that we use in (2.4) as well as the sequel is the ℓ_∞-norm. One of the advantages of using the ℓ_∞-norm is that the hypercubic neighborhood is easily computed. More generally, a hypercube is a special case of a polyhedron, which is important when we want to propagate the image of this set under linear dynamics, which, by exploiting the linearity of Σ, is easily done. This is useful in Section 2.4.4 when we describe our algorithm for reach set computation.

2.3 Assumptions and Notations

In this section, for clarity of exposition we briefly state the assumptions that are made and the notations that are used throughout this thesis.

[3] If we consider sets only of Euclidean space, then for two given sets $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^n$, it is defined as $d_H(X, Y) := \max\{\sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y)\}$ where $d(x, y) := \|x - y\|$.

2.3.1 Exact Computation Assumption

We begin our treatment by assuming that for a given LTI system Σ, a given initial state x_0, and a given time t, the vector $x(t) \in \mathbb{R}^n$ of an LTI system Σ can be computed with an arbitrarily small error. Subsequently, in the sequel, we show how the results can be generalized when it is possible to compute this vector with precision ϵ, for every $\epsilon > 0$. However, at the outset, we assume a capability to compute $x(t) = e^{At}x_0 + \int_0^t e^{A(t-s)}uds$ exactly.

We also assume that (i) the convex hull of a set of finite points in $\mathbb{R}^n$, and (ii) the intersection between a polyhedron and a hyperplane can be computed with an arbitrarily small error. These capabilities are used in the proposed algorithm, which we discuss in Sections 2.4.4 and 2.5.2, when it computes an over-approximation of a polyhedron and an approximation of a reachable state at the time that it makes a discrete transition, respectively.

2.3.2 Notation

In the sequel, we use $\mathcal{D}_t(\mathcal{P})$ to denote the set of states reached at time t from a set $\mathcal{P}$ at time 0. We also use $\mathcal{D}_t(\mathcal{P}, \gamma)$ to denote an over-approximation of $\mathcal{D}_t(\mathcal{P})$ with an approximation parameter $\gamma > 0$, calling it a γ-approximation of $\mathcal{D}_t(\mathcal{P})$ if it satisfies (i) $\mathcal{D}_t(\mathcal{P}) \subset \mathcal{D}_t(\mathcal{P}, \gamma)$ and (ii) $d_H(\mathcal{D}_t(\mathcal{P}), \mathcal{D}_t(\mathcal{P}, \gamma)) \leq \gamma$. Note that $\mathcal{D}_0(\mathcal{P}, \gamma)$ is simply a γ-approximation of the set $\mathcal{P}$. We will also use the notation $\mathcal{B}_r(x)$ to denote a closed ball of radius r with center x, i.e., $\mathcal{B}_r(x) := \{y \in \mathbb{R}^n : \|y - x\| \leq r\}$. Note that since we are using the ℓ_∞-norm, this is a hypercubic neighborhood, as shown in Figure 2.3.

2.4 Bounded ϵ-Reach Set of a DTLHA with Single Location

In this section, we consider the problem of bounded ϵ-reach set computation of the special case of a DTLHA $\mathcal{A}$ for which $\mathbb{L} = \{l_0\}$ and $Inv_0 = \mathcal{X}$. This problem is simply equivalent to bounded ϵ-reach set computation of a linear time-invariant (LTI) system Σ over a closed and bounded continuous domain $\mathcal{X}$ (that is the union of a finite number of polyhedra). More precisely, for a given fixed $(\Sigma, \mathcal{X}, T, x_0, \epsilon)$, we show that it is possible to compute a bounded

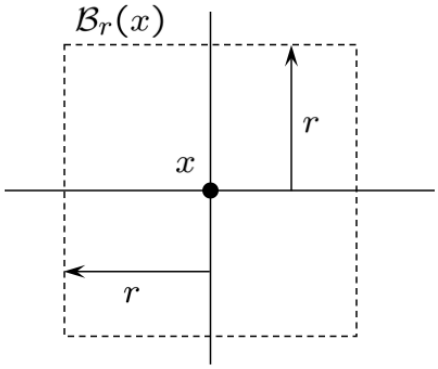


Figure 2.3: A closed ball, with respect to the ℓ_∞-norm, of radius r with center x, i.e., $\mathcal{B}_r(x)$.

ϵ-reach set of Σ from an initial state x_0 up to time t_f, where $t_f := \min\{\tau_1, T\}$ with $\tau_1 := \max_{t\in[0,T]}\{t : x(\tau) \in \mathcal{X},\ \forall\tau \in [0,t]\}$.

2.4.1 Bounded ϵ-Reach Set of an LTI System from a Fixed Initial State

We first describe the idea of computing a bounded ϵ-reach set of Σ from a fixed initial state x_0 through sampling with a period h. As shown in Figure 2.4, to over-approximate a trajectory via a finite set of sampled states obtained by sampling at a finite set of times, we first need to determine a neighborhood centered at each sampled state in which the trajectory between sampled states is guaranteed to be contained. We now show that for a suitable value of r, dependent on the sampling interval, $\mathcal{B}_r(x(t))$ constitutes such a neighborhood at a state $x(t)$ sampled by a sampling period h.

Suppose that we are specified an over-approximation bound ϵ. We wish to determine a sampling period h which guarantees that $x(\tau) \in \mathcal{B}_\epsilon(x(t))$ for $\tau \in [t, t+h]$. Hence we wish to ensure that

$$\max_{\tau\in[0,h]} \|x(t+\tau) - x(t)\|_\infty < \epsilon \qquad \forall x(t) \in \mathcal{X}. \tag{2.5}$$

Now we determine a suitable value of h which achieves (2.5).

For a given Σ, $\mathcal{X}$, and $x(s) \in \mathcal{X}$, we have

$$\begin{aligned} \max_{s\in[t,t+\tau]} \|\dot{x}(s)\|_\infty &= \max_{s\in[t,t+\tau]} \|Ax(s)+u\|_\infty \\ &\leq \max_{s\in[t,t+\tau]} \{\|A\|_\infty \|x(s)\|_\infty + \|u\|_\infty\} \\ &\leq \|A\|_\infty \bar{x} + \|u\|_\infty, \end{aligned} \tag{2.6}$$

where $\bar{x} = \max_{x\in\mathcal{X}} \|x\|_\infty$.

Then for a fixed τ, we can compute an upper bound on $\|x(t+\tau)-x(t)\|_\infty$ as follows:

$$\begin{aligned} \|x(t+\tau)-x(t)\|_\infty &\leq \int_t^{t+\tau} \|\dot{x}(s)\|_\infty ds \\ &\leq \int_t^{t+\tau} \max_{s\in[t,t+\tau]} \|\dot{x}(s)\|_\infty ds \\ &\leq \int_t^{t+\tau} (\|A\|_\infty \bar{x} + \|u\|_\infty) ds \\ &= (\|A\|_\infty \bar{x} + \|u\|_\infty)\tau. \end{aligned} \tag{2.7}$$

Maximization of both sides of (2.7) over $\tau \in [0,h]$ gives us

$$\max_{\tau\in[0,h]} \|x(t+\tau)-x(t)\|_\infty \leq (\|A\|_\infty \bar{x} + \|u\|_\infty)h. \tag{2.8}$$

If we need to upper bound the right hand side by $\epsilon > 0$, then we can choose

$$h < \frac{\epsilon}{\|A\|_\infty \bar{x} + \|u\|_\infty}. \tag{2.9}$$

So, if we choose h as

$$h = \frac{\epsilon/2}{\|A\|_\infty \bar{x} + \|u\|_\infty}, \tag{2.10}$$

then it is clear that we can ensure (2.5). We note for future reference that the factor of half in the denominator is not necessary; its additional purpose will be clearer in the sequel.

Lemma 1. *Given $\epsilon > 0$, a bounded ϵ-reach set $\mathcal{R}_T(x_0,\epsilon)$ of an LTI system Σ from an initial state x_0 for time $[0,T]$ can be determined as follows:*

$$\mathcal{R}_T(x_0,\epsilon) := \bigcup_{k=0}^{m-1} \mathcal{B}_\epsilon(x(kh)), \tag{2.11}$$

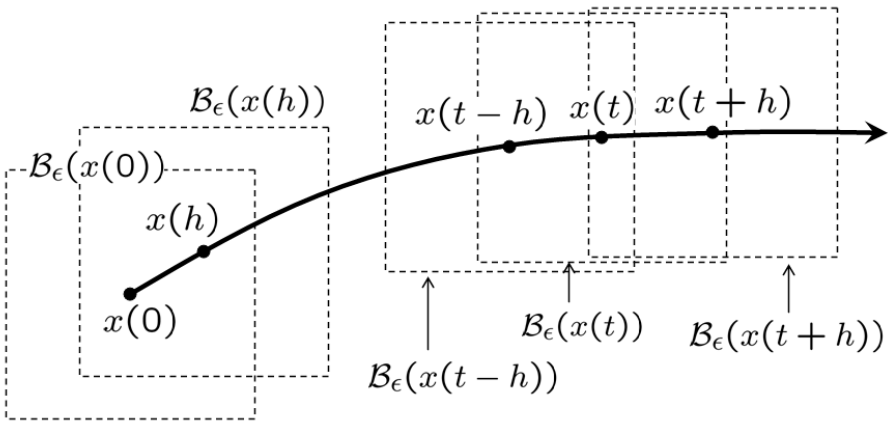


Figure 2.4: An over-approximation of a trajectory $x(t)$ through sampling.

where $m := \lceil T/h \rceil$ *and* $h = (\epsilon/2)/(\|A\|_\infty \bar{x} + \|u\|_\infty)$. *Moreover, this set has two additional properties:*

(i) $\lim_{\epsilon \to 0} \mathcal{R}_T(x_0, \epsilon) = \mathcal{R}_T(x_0)$, *and*

(ii) It contains an $\epsilon/2$ *neighborhood of* $\mathcal{R}_T(x_0)$, *i.e.,*

$$\bigcup_{z \in \mathcal{R}_T(x_0)} \mathcal{B}_{\epsilon/2}(z) \subseteq \mathcal{R}_T(x_0, \epsilon).$$

Proof. From (2.10), it is easy to see that $\cup_{k=0}^{m-1} \mathcal{B}_\epsilon(x(kh))$ is a bounded ϵ-reach set of Σ from x_0 for the given ϵ. Next, by the relation between ϵ and h in (2.10), it is clear that $h \to 0$ as $\epsilon \to 0$. This implies that $\mathcal{R}_T(x_0, \epsilon) \to \mathcal{R}_T(x_0)$ as $\epsilon \to 0$, establishing (i). For (ii), as noted above, (2.10) actually chooses half the sampling period that would have sufficed to make it a bounded ϵ-reach set over $[0, T]$. Hence, replacing ϵ by $\epsilon/2$ in the right hand side of (2.11) still yields a bounded ϵ-reach set. Thus the overstringent choice of h contains not just $\mathcal{R}_T(x_0)$ but actually all points that are within a distance $\epsilon/2$ from it. □

In summary, the bounded reach set $\mathcal{R}_T(x_0)$ can be over-approximated with arbitrarily small approximation error through sampling and polyhedral (actually hypercubic) over-approximation of each sampled state.

2.4.2 Continuity Property of a Bounded ϵ-Reach Set of an LTI System

Now we consider the situation where the initial condition can be in a small ball $\mathcal{B}_\delta(x_0)$ around x_0. We will extend the approach explained in Section 2.4.1 to an approach which computes a bounded ϵ-reach set of Σ from an initial set $\mathcal{B}_\delta(x_0)$ that is a small neighborhood centered at x_0.

We first show that there is a $\delta \in \mathbb{R}^+$ such that the bounded reach set of an LTI system Σ from $\mathcal{B}_\delta(x_0)$ is contained in a bounded ϵ-reach set of Σ from x_0 defined in (2.11).

Lemma 2. *Given $\epsilon > 0$, an LTI system Σ, an initial state x_0, and a time interval $[0, T]$, there exists a $\delta > 0$ such that*

$$\mathcal{R}_T(\mathcal{B}_\delta(x_0)) \subseteq \mathcal{R}_T(x_0, \epsilon), \tag{2.12}$$

where $\mathcal{B}_\delta(x_0)$ is a δ-ball around x_0 and $\mathcal{R}_T(\mathcal{B}_\delta(x_0))$ is the bounded reach set of Σ from $\mathcal{B}_\delta(x_0)$ up to time T. In particular, $\mathcal{R}_T(\mathcal{B}_{\epsilon/(2C)}(x_0)) \subseteq \mathcal{R}_T(x_0, \epsilon)$ for an appropriate C.

Proof. Note that $x(t) = e^{At}x_0 + \int_0^t e^{A(t-s)}uds$. If we consider two different initial states x_0 and y_0 in $\mathcal{B}_\delta(x_0)$, then their trajectories $x(t)$ and $y(t)$ satisfy $x(t) - y(t) = e^{At}(x_0 - y_0)$. Hence $\|x(t) - y(t)\| \leq ce^{\lambda t}\|x_0 - y_0\|$ for some positive constant c and some constant λ. Let $C := c \cdot \max_{0 \leq t \leq T}\{e^{\lambda t}\}$. So

$$\|x(t) - y(t)\| \leq C\|x_0 - y_0\| \qquad \text{for} \quad t \in [0, T]. \tag{2.13}$$

Since $\|x_0 - y_0\| \leq \delta$, $\|x(t) - y(t)\| \leq C\delta$ for all $t \in [0, T]$. In particular, this also implies that any initial condition y_0 in $\mathcal{B}_\delta(x_0)$ results in a $y(t)$ that lies in a $C\delta$ neighborhood of $\mathcal{R}_T(x_0)$ for all $t \in [0, T]$. Hence from property (ii) of Lemma 1, it also follows that $\mathcal{R}_T(\mathcal{B}_\delta(x_0)) \subseteq \mathcal{R}_T(x_0, 2C\delta)$. If we set $\delta = \epsilon/(2C)$, then it is clear that $\mathcal{R}_T(\mathcal{B}_\delta(x_0)) \subseteq \mathcal{R}_T(x_0, \epsilon)$. □

From the computational point of view, computing $\mathcal{R}_T(\mathcal{B}_\delta(x_0))$ is not known to be decidable. However, we can over-approximate it since we can compute $\mathcal{R}_T(x_0, \epsilon)$ as stated in Lemma 1. This is important for the development of our algorithm for a bounded ϵ-reach set computation.

Next we extend the result in Lemma 2 from the bounded reach set $\mathcal{R}_T(\mathcal{B}_\delta(x_0))$ to an over-approximation of it, denoted as $\mathcal{R}_T(\mathcal{B}_\delta(x_0), \gamma)$.

Lemma 3. *Given $\epsilon > 0$, an LTI system Σ, an initial state x_0, and a time interval $[0, T]$, there exist $\delta > 0$ and $\gamma > 0$ such that $\mathcal{R}_T(\mathcal{B}_\delta(x_0), \gamma) \subseteq \mathcal{R}_T(x_0, \epsilon)$. In particular, $\mathcal{R}_T(x_0) \subseteq \mathcal{R}_T(\mathcal{B}_{\epsilon/(4C)}(x_0), \epsilon/4) \subseteq \mathcal{R}_T(x_0, \epsilon)$.*

Proof. Let $x(t; z)$ denotes the solution at time t of the differential equation $\dot{x}(t) = Ax(t) + u$ with initial condition $x(0) = z \in \mathcal{B}_\delta(x_0)$. Now consider $w \in \mathcal{R}_T(\mathcal{B}_\delta(x_0), \gamma)$. Then $\|w - x(t; z)\| < \gamma$ for some $t \in [0, T]$ and $z \in \mathcal{B}_\delta(x_0)$. Hence

$$\begin{aligned} \|w - x(t; x_0)\| &= \|w - x(t; z) + x(t; z) - x(t; x_0)\| \\ &\leq \|w - x(t; z)\| + \|x(t; z) - x(t; x_0)\| \\ &\leq \gamma + \|x(t; z) - x(t; x_0)\|. \end{aligned}$$

From (2.13), we know that $\|x(t; z) - x(t; x_0)\| \leq C\|z - x_0\| \leq C\delta$. Therefore w lies in a $(\gamma + C\delta)$-neighborhood of $\mathcal{R}_T(x_0)$. Hence

$$\|w - x(t; x_0)\| \leq \gamma + C\delta.$$

From the property (ii) of the over-approximation in Lemma 1, we have $w \in \mathcal{R}_T(x_0, 2(\gamma + C\delta))$. Hence we see that $\mathcal{R}_T(\mathcal{B}_\delta(x_0), \gamma) \subseteq \mathcal{R}_T(x_0, 2(\gamma + C\delta))$. So, given $\epsilon > 0$, we can choose $\gamma = \epsilon/4$ and $\delta = \epsilon/(4C)$, and then $\mathcal{R}_T(\mathcal{B}_\delta(x_0), \gamma) \subseteq \mathcal{R}_T(x_0, \epsilon)$. □

2.4.3 Decidability of Exit Strictly Before T for an LTI System

We continue to consider an LTI system as in the previous section. However, we focus now on the time τ_1 that it exits $\mathcal{X}$. We note that it is not known to be decidable to compute τ_1 exactly. However, we show below that if the state $x(t)$ of the LTI system Σ from an initial state x_0 first crosses the boundary of the given domain $\mathcal{X}$ at time $\tau_1 < T$, then that fact can be determined by a suitable over-approximation.

Lemma 4. *Given an LTI system Σ, an initial state x_0 in the given compact domain $\mathcal{X}$, and a time interval $[0, T]$, if $\tau_1 < T$, then for all small enough $\delta > 0$ and for some small enough $h > 0$, $\mathcal{B}_\delta(x(nh)) \subset \mathcal{X}^C$ for some $n \in \mathbb{N}$.*

Proof. Let $\vec{n}_1$ be an outward normal vector of $\partial\mathcal{X}$ at $x(\tau_1)$. Since $\langle \dot{x}(\tau_1), \vec{n}_1 \rangle > 0$ by assumption, then by the continuity of the vector field of Σ, there exists

an $r > 0$ such that for all $x(t) \in \mathcal{B}_r(x(\tau_1)) \cap \partial\mathcal{X}$, $\langle \dot{x}(t), \vec{n}_1 \rangle > 0$. If we let $\bar{v} := \|A\|_\infty \bar{x} + \|u\|_\infty$ for a given LTI system Σ, where $\bar{x} = 2\max_{x\in\mathcal{X}} \|x\| + r$, we moreover have $\|\dot{x}(t)\| \leq \bar{v}$. (We note that this value of $\bar{x}$ is larger than the value necessary for (2.10), and so, in the sequel, we will adopt this larger value for $\bar{x}$). Hence $x(t) \in \mathcal{X}^C$ for $t \in (\tau_1, \tau_1 + 2h)$ for any $h > 0$ such that $2h < r/\bar{v}$. This implies that $x(nh) \in \mathcal{X}^C$ for some n. Moreover by compactness of $\mathcal{X}$, there exists a $\delta > 0$ such that $\mathcal{B}_\delta(x(nh)) \subset \mathcal{X}^C$. □

Now suppose that $x(t) \in \mathcal{X}$ for all $0 \leq t \leq T + \theta$ for some $\theta > 0$. Then that fact can also be determined.

Lemma 5. *Suppose $x(t) \in \mathcal{X}$ for all $0 \leq t \leq T + \theta$ for some $\theta > 0$. Then for all small enough $\delta > 0$ and $\gamma > 0$,*

$$\mathcal{R}_T(\mathcal{B}_\delta(x_0), \gamma) \subseteq \mathcal{X}^\circ. \tag{2.14}$$

Proof. Since $x(t) \in \mathcal{X}^\circ$ for all $0 \leq t \leq T$, the result follows. □

From the above, we see that even though determining that $\tau_1 = T$ *exactly* is not known to be decidable, nevertheless, if $\tau_1 < T$ or $\tau_1 > T$, then those facts can be determined computationally.

2.4.4 Algorithm for a Bounded ϵ-Reach Set of an LTI System from a Neighborhood of an Initial Condition

In this section, we propose an algorithm for a bounded ϵ-reach set of an LTI system from a δ-neighborhood of a given initial state x_0 based on the theoretical result presented in Sections 2.4.1, 2.4.2, and 2.4.3. Notice that this presented algorithm will be extended later when we discuss an algorithm for a bounded ϵ-reach set computation of a general case of DTLHA from an initial state x_0.

Let polyhedron $\mathcal{P}_0$ denote a δ-neighborhood of x_0, as shown in Figure 2.5. A special case of such a polyhedron is $\mathcal{B}_\delta(x_0)$. Note that $\mathcal{D}_t(\mathcal{P}_0)$ denotes the set of reachable states from $\mathcal{P}_0$ at time t under the given dynamics, as mentioned in Section 2.3.2. Since we consider an LTI system in this section, $\mathcal{D}_t(\mathcal{P}_0)$ is simply the image of $\mathcal{P}_0$ under the linear dynamics Σ, i.e, $\mathcal{D}_t(\mathcal{P}_0) := \{e^{At}x + \int_0^t e^{As}uds : x \in \mathcal{P}_0\}$. At each time t, an over-approximation

of $\mathcal{D}_t(\mathcal{P}_0)$ will be denoted by $\mathcal{D}_t(\mathcal{P}_0, \gamma)$, where γ is an over-approximation parameter. That is, for any $y \in \mathcal{D}_t(\mathcal{P}_0, \gamma)$,

$$\exists z \in \mathcal{D}_t(\mathcal{P}_0) \quad \text{s.t.} \quad \|y - z\| \leq \gamma. \tag{2.15}$$

The proposed algorithm is shown in Algorithm 1. In computing a bounded ϵ-reach set for a given $\epsilon > 0$, two variables identified in Lemma 3, γ and δ, will be critical. These variables will be iteratively reduced (or updated in the more general case) in the function `ReachSet`$(\cdot)$, through being multiplied by a given fixed parameter α that can take a value in the range $(0, 1)$ until the algorithm terminates. At each iteration of the **while** loop, a sampling period h satisfying the over-approximation condition is determined by $h = (\gamma/2)/(\|A\|_\infty \bar{x} + \|u\|_\infty)$, as shown in (2.10) for a current value of γ.

The algorithm terminates only when the function `ReachSet`$(\cdot)$ returns a nonempty set $\mathcal{R}$ that is computed as a bounded ϵ-reach set in `ReachSet`$(\cdot)$ for the given input. If an empty $\mathcal{R}$ is returned, then this implies that `ReachSet`$(\cdot)$ has failed its computation and this triggers the algorithm to restart its computation from the initial state x_0 at time 0 with different values of δ and γ.

For the given input of the algorithm, $(\Sigma, \mathcal{X}, T, x_0, \epsilon, \alpha)$, and the values for the parameters δ, γ, and h, the function `ReachSet`$(\cdot)$ computes a bounded ϵ-reach set of Σ from x_0 up to $t_f := \min\{\tau_1, T\}$ and returns a set of outputs including the updated γ and δ, a computed bounded ϵ-reach set $\mathcal{R}$, the time t when `ReachSet`$(\cdot)$ terminates, and the set $\mathcal{P}$ that is the image of Σ from $\mathcal{P}_0$ at time t. The function `ReachSet`$(\cdot)$ does this by starting its computation from the given $\mathcal{P}_0$ at time t_0, continuing until either the condition $t \leq T$ or the condition $\mathcal{D}_t(\mathcal{P}_0) \subset \mathcal{X}$ is not satisfied. If the $t \leq T$ condition is not satisfied, then the returned $\mathcal{R}$ is a nonempty set; it is in fact $\mathcal{R}_T(x_0, \epsilon)$. If however, the condition $\mathcal{D}_t(\mathcal{P}_0) \subset \mathcal{X}$ is violated, there are two possibilities. The first case is when $\mathcal{D}_t(\mathcal{P}_0) \subset \mathcal{X}^C$, i.e., $\tau_1 < T$. For this case, the returned value of $\mathcal{P}$ is set equal to $\mathcal{D}_t(\mathcal{P}_0)$, and a value of t is returned which is such that $\tau_1 \in [t - h, t]$. Thus $\mathcal{R} = R_{\tau_1}(x_0, \epsilon)$. For the other case, `ReachSet`$(\cdot)$ first updates δ by multiplying by the given parameter α to shrink the initial neighborhood $\mathcal{B}_\delta(x_0)$, and returns an empty $\mathcal{R}$.

In `ReachSet`$(\cdot)$, $\mathcal{R}_{t_f}(x_0, \epsilon)$ is computed in the following way. Note that $\mathcal{P}_0$ is a polyhedron defined by an ℓ_∞ norm as shown in Figure 2.5. In fact,

Algorithm 1: A bounded ϵ-reach set computation for Σ from an initial state x_0 for some constant $\alpha \in (0, 1)$.

Input: $\Sigma, \mathcal{X}, T, x_0, \epsilon, \alpha$
Output: $\mathcal{R}$

$\gamma = \alpha\epsilon; \delta = \alpha\epsilon; \mathcal{R} = \emptyset$
while $\mathcal{R} = \emptyset$ **do**
  $t_0 = 0; \mathcal{P}_0 = \mathcal{B}_\delta(x_0)$
  $h = (\gamma/2)/(\|A\|\bar{x} + \|u\|)$
  $(\gamma, \delta, t, \mathcal{P}, \mathcal{R}) = \texttt{ReachSet}(\Sigma, \mathcal{X}, \mathcal{P}_0, T, t_0, h, \gamma, \delta, \epsilon, \alpha)$
end
return $\mathcal{R}$

`ReachSet`$(\cdot)$ exploits this polyhedral structure to compute an image $\mathcal{D}_{t+h}(\mathcal{P}_0)$ from $\mathcal{D}_t(\mathcal{P}_0)$ at each time t. If we let $\mathcal{V}_t$ be the set of vertices of $\mathcal{D}_t(\mathcal{P}_0)$, then for each $x_i(t) \in \mathcal{V}_t$, $x_i(t+h)$ can be computed as follows:

$$x_i(t+h) = e^{Ah}x_i(t) + \int_0^h e^{A(h-\tau)}u \; d\tau, \tag{2.16}$$

where A and u are given by the system Σ. If we denote by $\mathcal{V}_{t+h}$ the set of $x_i(t+h)$ computed from $\mathcal{V}_t$, then it is easy to see that $\mathcal{D}_{t+h}(\mathcal{P}_0)$ is the polyhedral convex hull of $\mathcal{V}_{t+h}$. Next, a polyhedron $\mathcal{D}_t(\mathcal{P}_0, \gamma)$ is constructed to over-approximate $\mathcal{D}_\tau(\mathcal{P}_0)$ for $\tau \in [t, t+h]$. Note that if h satisfies the condition $h < \gamma/(\|A\|_\infty \bar{x} + \|u\|_\infty)$ for a given γ, then $\mathcal{D}_t(\mathcal{P}_0, \gamma)$ is guaranteed to be an over-approximation of $\mathcal{D}_\tau(\mathcal{P}_0)$ for $\tau \in [t, t+h]$. For each $x_i(t) \in \mathcal{V}_t$, if we regard $\mathcal{B}_\gamma(x_i(t))$ as an over-approximation of the trajectory $x_i(\tau)$ for $\tau \in [t, t+h]$ with respect to a given γ, then $\mathcal{D}_t(\mathcal{P}_0, \gamma)$ can be constructed as a polyhedral convex hull of $\cup_{x_i(t) \in \mathcal{V}_t} \mathcal{V}(\mathcal{B}_\gamma(x_i(t)))$ where $\mathcal{V}(\mathcal{B}_\gamma(x_i(t)))$ is a set of vertices of the polyhedron $\mathcal{B}_\gamma(x_i(t))$. This process of image computation is illustrated in Figure 2.5.

Lemma 6. *Let $\mathcal{H}$ be the convex hull of $\cup_{v_i \in \mathcal{V}_t} \mathcal{V}(\mathcal{B}_\gamma(v_i))$. Then $\mathcal{H}$ is exactly the closed γ-neighborhood of the convex hull of $\mathcal{V}_t$.*

Proof. Note that if $\bar{w} \in \mathcal{H}$, then $\bar{w} = \lambda\bar{y}_1 + (1-\lambda)\bar{y}_2$ where $\|\bar{y}_1 - v_1\| = \gamma$

and $\|\bar{y}_2 - v_2\| = \gamma$ for some $v_1, v_2 \in \mathcal{V}_t$ and $0 \leq \lambda \leq 1$. Then

$$\begin{aligned} \|\bar{w} - (\lambda v_1 + (1-\lambda)v_2)\| &= \|\lambda(\bar{y}_1 - v_1) + (1-\lambda)(\bar{y}_2 - v_2)\| \\ &\leq \lambda\|\bar{y}_1 - v_1\| + (1-\lambda)\|\bar{y}_2 - v_2\| \\ &\leq \gamma. \end{aligned} \tag{2.17}$$

Hence $\bar{w}$ belongs to the γ-neighborhood of the convex hull of $\mathcal{V}_t$.

For the converse, consider $\bar{z}$ in the γ-neighborhood of the convex hull of $\mathcal{V}_t$. Then for some $\lambda_i \geq 0$, $\sum_i \lambda_i = 1$, $\|\bar{z} - \sum_i \lambda_i v_i\| \leq \gamma$, where $v_i \in \mathcal{V}_t$. Let $s := \bar{z} - \sum_i \lambda_i v_i$. Now $\bar{z} = \sum_i \lambda_i (v_i + s)$. So $\bar{z}$ is in the convex hull of $\{v_i + s\}$. However each $v_i + s \in \mathcal{B}_\gamma(v_i)$. Hence each $v_i + s$ is in the convex hull of the vertices of $\mathcal{B}_\gamma(v_i)$. Thus $\bar{z}$ is in $\mathcal{H}$. □

If the diameter of $\mathcal{D}_t(\mathcal{P}_0, \gamma)$ is less than the given ϵ, and $\mathcal{D}_t(\mathcal{P}_0)$ is still contained in $\mathcal{X}$, then the algorithm takes $\mathcal{D}_t(\mathcal{P}_0, \gamma)$ as a valid ϵ-reach set at time t. Otherwise, `ReachSet`$(\cdot)$ first updates γ and δ by a given parameter α to shrink the size of $\mathcal{D}_t(\mathcal{P}_0, \gamma)$, and returns an empty $\mathcal{R}$ to restart its computation from a smaller $\mathcal{P}_0$ and h.

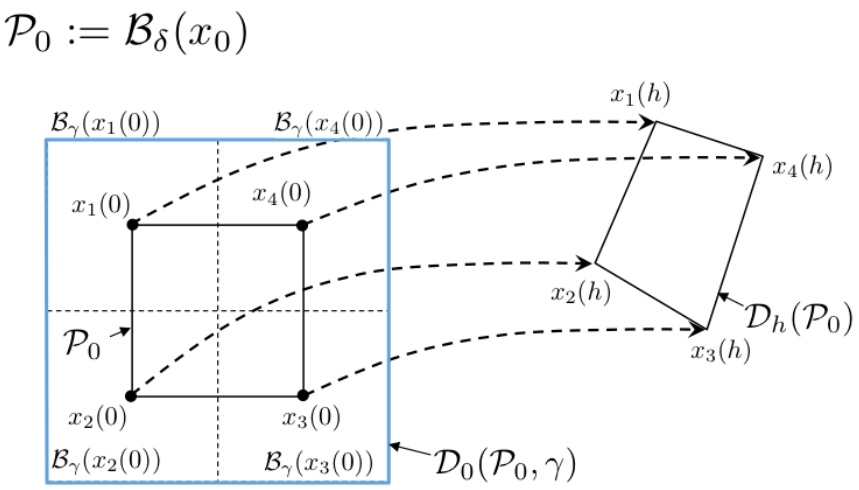
Now, we show that $\mathcal{R}$ computed by the Algorithm 1 is indeed a bounded ϵ-reach set.

Theorem 1. *Given input $(\Sigma, T, \mathcal{X}, x_0, \epsilon)$, Algorithm 1 terminates in a finite number of iterations and returns a bounded ϵ-reach set of Σ from x_0 up to time* $\min\{\tau_1, T\}$.

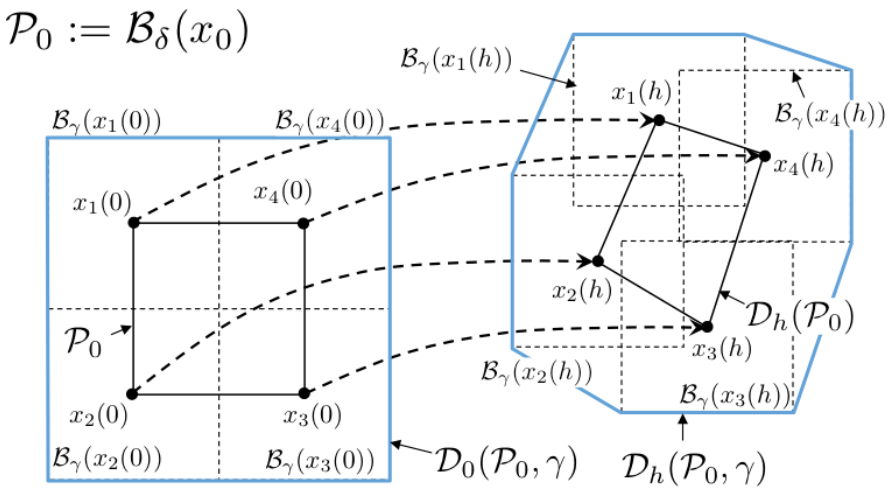
Proof. If $\tau_1 < T$, then by Lemma 4, there exists an $r > 0$ such that $\mathcal{B}_r(x(nh)) \subset \mathcal{X}^C$ for some n. Let $t_a := nh$. Then for this r, Lemma 3 shows that there exist $\delta_1 > 0$ and $\gamma_1 > 0$ such that $\mathcal{D}_{t_a}(\mathcal{B}_{\delta_1}(x_0), \gamma_1) \subseteq \mathcal{B}_r(x(t_a))$

Let $t_f := \min\{\tau_1, T\}$. Then for a given $\epsilon > 0$, we can find $\delta_2 > 0$ and $\gamma_2 > 0$ such that $\mathcal{R}_{t_f}(\mathcal{B}_{\delta_2}(x_0), \gamma_2) \subseteq \mathcal{R}_{t_f}(x_0, \epsilon)$ by Lemma 3. Now let $\delta := \min\{\delta_1, \delta_2\}$ and $\gamma := \min\{\gamma_1, \gamma_2\}$. Then since $\delta > 0$ and $\gamma > 0$, the Algorithm 1 terminates in a finite number of iterations and returns $\mathcal{R}$. Since $\delta < \delta_1$, $t_f = \tau_1$ if $\tau_1 < T$. Otherwise, $t_f = T$. Moreover, $\delta < \delta_2, \gamma < \gamma_2$ implies that $\mathcal{R}$ is an ϵ-reach set of Σ from x_0 up to time t_f. □

We now have the following theorem which says that even though Algorithm 1 may not determine τ_1 exactly, it can determine an estimate of τ_1 with any desired accuracy, by using a sufficiently small $\epsilon > 0$.



(a) Computation of $\mathcal{D}_h(\mathcal{P}_0)$, the image of $\mathcal{P}_0$ at time h.



(b) Computation of $\mathcal{D}_h(\mathcal{P}_0, \gamma)$ from $\mathcal{D}_h(\mathcal{P}_0)$.

Figure 2.5: The image computation in `ReachSet`$(\cdot)$.

Theorem 2. *If $\tau_1 < T$, then for any arbitrarily small $\rho > 0$, there exists $\epsilon > 0$ such that $|\tau_1 - t_1| \leq \rho$ where t_1 is the time when Algorithm 1 terminates.*

Proof. Let γ_ϵ be the γ when Algorithm 1 terminates for a given ϵ. We also let h_ϵ be a sampling period corresponding to γ_ϵ. Hence $\tau_1 \in [t_1 - h_\epsilon, t_1]$. Since $h_\epsilon \to 0$ as $\gamma_\epsilon \to 0$ and $\gamma_\epsilon \to 0$ as $\epsilon \to 0$, $h_\epsilon \to 0$ as $\epsilon \to 0$. Therefore, there exists $\epsilon > 0$ such that $h_\epsilon \leq \rho$. □

2.5 Bounded ϵ-Reach Set of a DTLHA with Multiple Locations

In this section, we consider the problem of bounded ϵ-reach set computation of the general case of a DTLHA $\mathcal{A}$ whose $\mathbb{L}$ is not a singleton and $\cup_{l \in \mathbb{L}} Inv(l) = \mathcal{X}$ (thus differentiating it from the case that is considered in Section 2.4). We first show that, even though it is not known to be decidable to determine the discrete transition state and time exactly, a discrete transition can be determined computationally through an over-approximation of the discrete transition state and time if the discrete transition is deterministic and transversal. Then we extend Algorithm 1 proposed in Section 2.4.4 to compute a bounded ϵ-reach set of a DTLHA $\mathcal{A}$. More precisely, for a given fixed input $(\mathcal{A}, \mathcal{X}, N, T, l_0, x_0, \epsilon)$, the proposed algorithm computes a bounded ϵ-reach set $\mathcal{R}_{t_f}(x_0, \epsilon)$ of an LHA $\mathcal{A}$ over a compact domain $\mathcal{X}$ from an initial state $x_0 \in Inv_0$ under the assumption that every discrete transition up to time t_f is a deterministic and transversal discrete transition, where $t_f := \{\tau_N, T\}$, N is an upper bound on the number of discrete transitions, T is an upper bound on time, and τ_N is the time when the N-th discrete transition occurs.

2.5.1 Over-approximating the State and Time of a Discrete Transition

Given the first discrete transition at time τ_1, if $\tau_1 < T$ for a given initial state $x_0 \in Inv_0$, a bounded ϵ-reach set of $\mathcal{A}$ from x_0 up to time τ_1 can be computed as explained in Section 2.4.4. To continue the computation after τ_1 and until τ_2, it is necessary to know $x(\tau_1)$. However, it is not known to be decidable

to compute $x(\tau_1)$ exactly in general. Therefore, some over-approximation of $x(\tau_1)$ has to be computed to continue the ϵ-reach set computation beyond τ_1.

We now show that if $x(\tau_1)$ satisfies a deterministic and transversal discrete transition condition, then it is possible not only to determine that a discrete transition occurs at some time around τ_1 with arbitrarily fine precision, but also to compute an over-approximation of $x(\tau_1)$ with arbitrarily small approximation error.

Lemma 7. *Given $\tau_1 < T$, if $x(\tau_1) \in \partial Inv_0$ satisfies a deterministic and transversal discrete transition condition, then there exists a $\delta > 0$ such that $\mathcal{B}_{2\delta}(x(\tau_1)) \subset (Inv_0 \cup Inv_1)$ for some location l_1. Furthermore, there exists a $\Delta > 0$ such that*

(i) $x(t) \in (Inv_1)^\circ$ for $t \in (\tau_1, \tau_1 + \Delta)$, and

(ii) the following hold.

$$\bigcup_{y \in \mathcal{J}_{0,1}} \mathcal{R}_{(\tau_1, \tau_1+\Delta)}(y) \subset (Inv_1)^\circ, \tag{2.18}$$

where $\mathcal{R}_{(\tau_1,\tau_1+\Delta)}(y)$ is the reach set of $\mathcal{A}$ for time $(\tau_1, \tau_1 + \Delta)$ and $\mathcal{J}_{0,1} := \mathcal{B}_\delta(x(\tau_1)) \cap Inv_0 \cap Inv_1$.

Proof. Let Inv_1, Inv_2 be invariant sets for some locations l_1 and l_2 such that $Inv_0 \cap Inv_1 \cap Inv_2 \neq \emptyset$. Since $x(\tau_1)$ satisfies a deterministic discrete transition condition, if $x(\tau_1) \in Inv_0 \cap Inv_1$, then $x(\tau_1) \notin Inv_0 \cap Inv_2$. This implies that $x(\tau_1) \notin Inv_2$. Then by compactness of Inv_2, we know that there exists a $\delta' > 0$ such that $\mathcal{B}_{\delta'}(x(\tau_1)) \cap Inv_2 = \emptyset$. Therefore, we conclude that $\mathcal{B}_{\delta'}(x(\tau_1)) \subset Inv_0 \cup Inv_1$.

Let $\vec{n}_1$ be an outward normal vector of ∂Inv_0 at $x(\tau_1)$. Since $x(\tau_1)$ satisfies a transversal discrete transition condition from the location l_0 to the other location l_1, we know that there exists a $\delta'' > 0$ such that for all $x(t) \in \mathcal{B}_{\delta''}(x(\tau_1)) \cap Inv_0 \cap Inv_1$, $\langle \dot{x}(t), \vec{n}_1 \rangle > 0$, where $\dot{x}(t)$ is taken as either $A_0 x(t) + u_0$ or as $A_1 x(t) + u_1$, by the continuity of vector fields of Σ_0 and Σ_1.

Let $\delta = \min\{\delta'/2, \delta''/2\}$, and $\Delta := \delta/(2\bar{v})$ where $\bar{v} := \sup_{x \in \mathcal{J}_{0,1}} \max\{\|A_0\|\|x\| + \|u_0\|, \|A_1\|\|x\| + \|u_1\|\}$ and $\mathcal{J}_{0,1} := \mathcal{B}_\delta(x(\tau_1)) \cap Inv_0 \cap Inv_1$. Then (i) and (ii) hold for these choice of δ and Δ. □

In Lemma 7, $\mathcal{J}_{0,1}$ is an over-approximation of $x(\tau_1)$ that is determined by taking a δ-ball around $x(\tau_1)$ for suitably small $\delta > 0$, and intersecting it with Inv_0 and Inv_1. Once such a suitably small δ is known, then the following lemma shows that it is also possible to determine a δ_0-neighborhood of an initial state x_0 such that the reach set at time τ_1 of $\mathcal{A}$, from the δ_0-neighborhood of x_0, is contained in an δ-ball of $x(\tau_1)$.

Lemma 8. *Given δ determined by Lemma 7, there exists a δ_0 such that*

$$\mathcal{D}_{\tau_1}(\mathcal{B}_{\delta_0}(x_0)) \subseteq \mathcal{B}_{\delta}(x(\tau_1)), \tag{2.19}$$

and $\mathcal{D}_{\tau_1}(\mathcal{B}_{\delta_0}(x_0)) \cap Inv_0 \cap Inv_1$ is an over-approximation of $x(\tau_1)$.

Proof. This lemma follows from the same argument used in the proof of Lemma 2, by choosing $\delta_0 = \delta/C$. □

Lemma 8 implies that no matter how small a δ-ball of $x(\tau_1)$ that is contained in $Inv_0 \cap Inv_1$, there always exists a corresponding δ_0-ball of the initial state x_0 such that the reach set of $\mathcal{A}$ from $\mathcal{B}_{\delta_0}(x_0)$ at time τ_1 is contained in $\mathcal{B}_{\delta}(x(\tau_1))$. The next lemma shows that such a $\mathcal{B}_{\delta_0}(x_0)$ can be determined for any discrete transition.

Lemma 9. *Let δ_k be the radius of a ball centered at $x(\tau_k)$ intersecting only Inv_{k-1} and Inv_k, where τ_k is the k-th discrete transition time and l_k is the location after the k-th discrete transition. Then for any $x(\tau_k)$ satisfying a deterministic and transversal discrete transition condition, there exists a δ_0 such that*

$$\mathcal{D}_{\tau_k}(\mathcal{B}_{\delta_0}(x_0)) \subseteq \mathcal{B}_{\delta_k}(x(\tau_k)), \tag{2.20}$$

where $\mathcal{D}_{\tau_k}(\mathcal{B}_{\delta_0}(x_0))$ is the image of $\mathcal{B}_{\delta_0}(x_0)$ at time τ_k.

Proof. From the continuity property shown in Lemma 2, there is a $\delta_{k-1} > 0$ such that $\mathcal{R}_{[\tau_{k-1},\tau_k]}(\mathcal{B}_{\delta_{k-1}}(x(\tau_{k-1}))) \subseteq \mathcal{R}_{[\tau_{k-1},\tau_k]}(x_{\tau_{k-1}}, \delta_k)$ for a given δ_k. Then for this δ_{k-1}, it is clear that $\mathcal{D}_{\tau_k}(\mathcal{B}_{\delta_{k-1}}(x(\tau_{k-1}))) \subseteq \mathcal{B}_{\delta_k}(x(\tau_k))$. Using the same argument, we can find $\delta_{k-2}, \delta_{k-3}, \cdots, \delta_1$. Then from Lemma 8, we know that there exists a $\delta_0 > 0$ such that $\mathcal{D}_{\tau_1}(\mathcal{B}_{\delta_0}(x_0)) \subseteq \mathcal{B}_{\delta_1}(x(\tau_1))$. Since $\mathcal{D}_{\tau_2}(\mathcal{B}_{\delta_1}(x(\tau_1))) \subseteq \mathcal{B}_{\delta_2}(x(\tau_2))$, we have $\mathcal{D}_{\tau_2}(\mathcal{B}_{\delta_0}(x_0)) \subseteq \mathcal{B}_{\delta_2}(x(\tau_2))$. This relation holds for each τ_i where $i = 1, 2, \cdots, k$. Therefore, $\mathcal{D}_{\tau_k}(\mathcal{B}_{\delta_0}(x_0)) \subseteq \mathcal{B}_{\delta_k}(x(\tau_k))$.

□

2.5.2 Algorithm for a Bounded ϵ-Reach Set of a DTLHA

In this section, we extend the algorithm presented in Section 2.4.4 to solve the problem of bounded ϵ-reach set computation of a general class DTLHA $\mathcal{A}$ based on the theoretical results shown in Section 2.4 and 2.5.1.

The proposed algorithm is shown in Algorithm 2. As in Algorithm 1, the Algorithm 2 terminates only when a nonempty set $\mathcal{R}$ is returned from the inner **while** loop. When an empty set $\mathcal{R}$ is returned, the algorithm restarts its bounded ϵ-reach set computation from the given initial state $x_0 \in Inv_0$ at time $t = 0$ with different values for the parameters δ and γ. As explained in Section 2.4.4, these parameter values are updated in `ReachSet`$(\cdot)$ when this function encounters an ambiguous situation with current values for these parameters and tries to resolve the situation with different values for the parameters. Besides the function `ReachSet`$(\cdot)$, the values for these parameters can also be updated inside the function `Transition`$(\cdot)$ which computes an over-approximate discrete transition state at the time of discrete transition. Hence, in general, the inner **while** loop terminates its iteration and returns an empty $\mathcal{R}$ whenever the parameter values for δ and γ are updated. Otherwise, for a given time upper bound T and a given upper bound N on the total number of discrete transitions, Algorithm 2 continues to compute a bounded ϵ-reach set of $\mathcal{A}$ until either $t > T$ or `jump` $> N$ within the inner **while** loop.

ReachSet$(\cdot)$ For the given inputs, the computational procedure in the function `ReachSet`$(\cdot)$ is the same as explained in Section 2.4.4. However, there are slight changes in the input and output of the function `ReachSet`$(\cdot)$. Inputs Σ_c, Inv_c, and $\mathcal{P}_c$ for this function represent the LTI dynamics, an invariant set, and the set of initial states at a location $l_c \in \mathbb{L}$ on which a bounded ϵ-reach set is going to be computed, respectively. Moreover, the input time t is the time when a polyhedron $\mathcal{P}_c \subset (Inv_c)^\circ$ is reached from an initial set $\mathcal{B}_\delta(x_0)$. Hence, if $t - h \leq \tau_k \leq t$, then `ReachSet`$(\cdot)$ computes $\mathcal{R}_{[\tau_k, t']}(x_0, \epsilon)$ at a location l_c starting from a given initial polyhedron $\mathcal{P}_c$ where τ_k is the time at the k-th discrete transition for some $k \in \mathbb{N}$ and $t' := \min\{\tau_{k+1}, T\}$.

If the function `ReachSet`$(\cdot)$ returns an empty set $\mathcal{R}_c$, then the algorithm restarts its overall bounded ϵ-reach set computation from $\mathcal{B}_\delta(x_0)$ at time

Algorithm 2: Bounded ϵ-reach set computation of $\mathcal{A}$ from an initial state (l_0, x_0) for some constant $\alpha \in (0, 1)$.

Input: $\mathcal{A}, N, T, l_0, x_0, \epsilon, \alpha$
Output: $\mathcal{R}$

$\gamma = \alpha\epsilon; \delta = \alpha\epsilon; \mathcal{R} = \emptyset$
$\bar{v} = \max_{v_i}\{v_i : v_i = \|A_i\|\bar{x} + \|u_i\|,\ \forall l_i \in \mathbb{L}\}$
while $\mathcal{R} = \emptyset$ **do**
  $t = 0; \mathsf{jump} = 0; l_c = l_0; \mathcal{P}_c = \mathcal{B}_\delta(x_0);\ h = (\gamma/2)/\bar{v}$
  while $\mathsf{jump} < N$ **do**
    $(\gamma, \delta, t, \mathcal{P}_c, \mathcal{R}_c) = \texttt{ReachSet}(\Sigma_c, Inv_c, \mathcal{P}_c, T, t, h, \gamma, \delta, \epsilon, \alpha)$
    if $\mathcal{R}_c = \emptyset$ **then**
      $\mathcal{R} = \emptyset$
      break
    end
    else if $(\mathcal{R}_c \neq \emptyset) \wedge (t > T)$ **then**
      $\mathcal{R} = \mathcal{R} \cup \mathcal{R}_c$
      break
    else
      $(\gamma, \delta, l_c, \mathcal{P}_c) = \texttt{Transition}(\mathcal{A}, l_c, \mathcal{P}_c, h, \gamma, \delta)$
      if $\mathcal{P}_c = \emptyset$ **then**
        $\mathcal{R} = \emptyset$
        break
      else
        $\mathcal{R} = \mathcal{R} \cup \mathcal{R}_c$
        $\mathsf{jump} = \mathsf{jump} + 1$
      end
    end
  end
end
return $\mathcal{R}$

$t = 0$ with the updated parameters γ and δ. Otherwise, a returned nonempty $\mathcal{R}_c \subset Inv_c$ is considered to be a valid ϵ over-approximation of the reach set of $\mathcal{A}$ from $x_0 \in Inv_0$ for time $[\tau_k, t']$, i.e., $\mathcal{R}_c = \mathcal{R}_{[\tau_k, t']}(x_0, \epsilon)$. If the returned time $t' > T$, then it is clear that $\mathcal{R} = \mathcal{R}_T(x_0, \epsilon)$, and the algorithm terminates returning $\mathcal{R}$ as a bounded ϵ-reach set for the given input to the algorithm.

If the image of $\mathcal{B}_\delta(x_0)$ at time t', i.e., $\mathcal{D}_{t'}(\mathcal{B}_\delta(x_0))$, lies entirely outside the invariant set Inv_c of the current location l_c, then $\texttt{ReachSet}(\cdot)$ returns to indicate an event of discrete transition from location l_c to some other locations. In the proposed algorithm, a variable `jump` is used to keep track of the total number of discrete transitions up to time t. For this case, the

returned time t' is less than the given time upper bound T. If $t' < T$, then the algorithm first needs to determine a new location to which a discrete transition can occur from the current location l_c, in order to further continue its computation.

Transition$(\cdot)$ At each moment of discrete transition, the algorithm first determines a sufficiently small neighborhood of initial states in the new location from which to continue its ϵ-reach set computation after the discrete transition. For a given input $(\mathcal{A}, l_c, \mathcal{P}_c, h, \gamma, \delta)$, the function `Transition`$(\cdot)$ determines a new $\mathcal{P}_c$ and l_c based on the assumption of a deterministic and transversal discrete transition condition. If the current values of γ and δ are sufficiently small enough to satisfy the transition conditions, then `Transition`$(\cdot)$ returns a unique l_c and a valid initial set $\mathcal{P}_c \in (Inv_c)^\circ$ at a new location $l_c \in \mathbb{L}$. Otherwise, the function returns an empty $\mathcal{P}_c$ and updated γ and δ. Then the algorithm restarts its bounded ϵ-reach set computation from $\mathcal{B}_\delta(x_0)$ at time $t = 0$ with new γ and δ.

Theorem 3. *Given input $(\mathcal{A}, N, T, Inv_0, x_0, \epsilon)$, Algorithm 2 terminates in a finite number of iterations and returns a bounded ϵ-reach set of $\mathcal{A}$ from $x_0 \in Inv_0$ up to time $\min\{\tau_N, T\}$, if $x(\tau_k)$ satisfies a deterministic and transversal discrete transition condition for every $\tau_k \leq \min\{\tau_N, T\}$.*

Proof. Let $t_f := \min\{\tau_N, T\}$. By Lemma 9, there exists a $\delta_0 > 0$ such that $\mathcal{D}_{\tau_k}(\mathcal{B}_{\delta_0}(x_0)) \subseteq \mathcal{B}_{\delta_k}(x(\tau_k))$ for all $\tau_k \leq t_f$. Hence $\mathcal{R}_{[\tau_{k-1},\tau_k]}(\mathcal{B}_{\delta_0}(x_0)) \subseteq \mathcal{R}_{[\tau_{k-1},\tau_k]}(\mathcal{B}_{\delta_{k-1}}(x(\tau_{k-1})))$. Then, for a given $\epsilon > 0$, $\mathcal{R}_{[\tau_{k-1},\tau_k]}(\mathcal{B}_{\delta_0}(x_0)) \subseteq \mathcal{R}_{[\tau_{k-1},\tau_k]}(x(\tau_{k-1}), \epsilon)$ since $\delta_k < \epsilon$ for each k, and $\mathcal{R}_{[\tau_{k-1},\tau_k]}(\mathcal{B}_{\delta_{k-1}}(x(\tau_{k-1})))$ $\subseteq \mathcal{R}_{[\tau_{k-1},\tau_k]}(x(\tau_{k-1}), \delta_k)$ by Lemma 2. By the same reason, $\mathcal{R}_{[\tau_f,t_f]}(\mathcal{B}_{\delta_0}(x_0))$ $\subseteq \mathcal{R}_{[\tau_f,t_f]}(x(\tau_f), \epsilon)$, where τ_f denotes the largest $\tau_k \leq t_f$. Since this holds for all $\tau_k \leq t_f$, we conclude $\mathcal{R}_{t_f}(\mathcal{B}_{\delta_0}(x_0)) \subseteq \mathcal{R}_{t_f}(x_0, \epsilon)$. Now we see that $\mathcal{R}_{t_f}(\mathcal{B}_{\delta_0'}(x_0), \gamma) \subseteq \mathcal{R}_{t_f}(x_0, \epsilon)$ for some $\delta_0' \in (0, \delta_0)$ and $\gamma > 0$ by Lemma 3. Since $\delta_0' > 0$ and $\gamma > 0$, it is easy to see that Algorithm 2 terminates in a finite number of iterations, and the returned $\mathcal{R}$ is an ϵ-reach set of $\mathcal{A}$ from x_0 up to time t_f from the argument above. □

CHAPTER 3

COMPUTING BOUNDED ϵ-REACH SET OF A DTLHA WITH FINITE PRECISION COMPUTATIONS

In Chapter 2, a class of hybrid automata, called Deterministic Transversal Linear Hybrid Automata (DTLHA) is proposed. A new computational approach is also proposed to compute an over-approximation of the reach set, with arbitrarily small approximation error $\epsilon \in \mathbb{R}^+$, up to a finite time, from an initial state. Such a set is referred to as a bounded ϵ-reach set.

The class of DTLHA consists of linear systems with constant inputs (i.e., where the right hand sides of the differential equations consist of the superposition of a term that is linear in the state and a constant input), for which the linear dynamics as well as the constant input switch along the boundaries of polyhedra, and for which the discrete transitions involved are *deterministic* and *transversal* at each discrete transition time. Since the solutions of linear systems involve matrix exponentials, one however needs to carefully take into account the issue of numerical approximations. In this chapter, we address the problem of computing a bounded ϵ-reach set of a DTLHA with variable finite precision numerical schemes and show that one can still compute a bounded ϵ-reach set.

We first briefly summarize the theoretical results for bounded ϵ-reachability of a DTLHA under the assumption of infinite precision calculation presented in Chapter 2. We then derive a set of conditions that can be used to determine the event of a deterministic and transversal discrete transition in computing a bounded ϵ-reach set of a DTLHA, which is discussed in more detail in Chapter 4. In the last part of this chapter, these results are extended to show that a bounded ϵ-reach set of a DTLHA can be computed even without the capability for infinite precision calculation.

3.1 Bounded ϵ-Reachability of a DTLHA with Infinite Precision Calculations

The approach to compute a bounded ϵ-reach set of a DTLHA from an initial state x_0 proposed in Chapter 2 is to over-approximate the bounded reach set through sampling and polyhedral over-approximation. More precisely, for given parameters δ and γ, and a sampling period h, the bounded reach set of a DTLHA from x_0 up to time t_f is over-approximated by

$$\bigcup_{m=0}^{M} \mathcal{D}_{mh}(\mathcal{B}_\delta(x_0), \gamma) \tag{3.1}$$

where $\mathcal{B}_\delta(x_0)$ is a polyhedral δ-neighborhood of x_0, γ is a parameter that defines the size of over-approximation of $\mathcal{D}_\tau(\mathcal{B}_\delta(x_0))$ for $\tau \in [0, t_f]$, and $M := \lceil t_f/h \rceil$.

In this approach, the existence of appropriate values for parameters δ, γ, and h is in fact critical in computing a bounded ϵ-reach set of a DTLHA from x_0. In Chapter 2, we showed that for any given $\epsilon \in \mathbb{R}^+$, there exist values for these parameters such that the set in (3.1) is indeed a bounded ϵ-reach set of an LHA from x_0 if every discrete transition up to time t_f is deterministic and transversal.

We present the main results from Chapter 2 as follows:

Lemma 10. *Given $\gamma \in \mathbb{R}^+$, if a sampling period h satisfies the following inequality in (3.2), then $\mathcal{D}_\tau(\mathcal{B}_\delta(x_0)) \subset \mathcal{D}_t(\mathcal{B}_\delta(x_0), \gamma)$ for $\tau \in [t, t+h]$ for each sample time t:*

$$h < \frac{\gamma}{\bar{v}} \tag{3.2}$$

where $\bar{v} := \max_{l_i \in \mathbb{L}}\{\|A_i\|\bar{x} + \|u_i\|\}$ and $\bar{x} := \max_{x \in \mathcal{X}} \|x\|$.

Lemma 11. *Given $\epsilon > 0$, a DTLHA $\mathcal{A}$, an initial state $(l_0, x_0) \in \mathbb{L} \times \mathbb{R}^n$, and a time bound t_f, there exist $\delta \in \mathbb{R}^+$, $\gamma \in \mathbb{R}^+$, and $h \in \mathbb{R}^+$ such that the following hold:*

(i) $\mathcal{R}_{t_f}(x_0) \subset \bigcup_{m=0}^{M} \mathcal{D}_{mh}(\mathcal{B}_\delta(x_0), \gamma)$,

(ii) $dia(\mathcal{D}_{mh}(\mathcal{B}_\delta(x_0), \gamma)) < \epsilon \quad \forall m \in \{0, 1, \cdots, M\}$,

(iii) *Suppose* $x(\tau_k) \in \partial Inv_i$, $x(\tau_k) = \lim_{\tau \to \tau_k} x(\tau)$, *and* $\tau_k < t_f$ *where* $x(\tau) \in (Inv_i)^\circ \ \forall \tau \in (\tau_k - \eta, \tau_k)$ *for some location* $l_i \in \mathbb{L}$ *and constant* $\eta \in \mathbb{R}^+$. *Then there exist* δ *and* h *such that*

(a) $\mathcal{D}_{t-h}(\mathcal{B}_\delta(x_0)) \subset (Inv_i)^\circ$,

(b) $\mathcal{D}_t(\mathcal{B}_\delta(x_0)) \subset (Inv_i)^C$,

(c) $\tau_k \in (t-h, t)$, *and*

(d) $t < t_f$,

where $\mathcal{D}_\tau(\mathcal{B}_\delta(x_0))$ *is computed under the LTI dynamics of* $l_i \in \mathbb{L}$ $\forall \tau \in [t-h, t]$, *and*

(iv) *Suppose (iii) holds and* $x(\tau_k)$ *makes a discrete transition from a location* l_i *to some other location* $l_j \in \mathbb{L}$. *Then there exists* δ, γ, *and* h *such that*

(a) $h < \Delta$, *and*

(b) $(\mathcal{D}_{\tau_k}(\mathcal{B}_\delta(x_0), \gamma) \cap Inv_i \cap Inv_j) \subset \mathcal{J}_{i,j}$

for some appropriate $\delta' \in \mathbb{R}^+$ *and* $\Delta \in \mathbb{R}^+$ *satisfying* $\mathcal{B}_{2\delta'}(x(\tau_k)) \subset (Inv_i \cup Inv_j)$ *and*

$$\bigcup_{y \in \mathcal{J}_{i,j}} \mathcal{D}_\tau(y) \subset (Inv_j)^\circ \quad \forall \tau \in (\tau_k, \tau_k + \Delta), \tag{3.3}$$

where $\mathcal{J}_{i,j} := \mathcal{B}_{\delta'}(x(\tau_k)) \cap Inv_i \cap Inv_j$.

where $\mathcal{R}_{t_f}(x_0)$ *is the bounded reach set of* $\mathcal{A}$, h *is determined by (3.2),* $dia(\mathcal{P})$ *denotes the diameter of a polyhedron* $\mathcal{P}$, *and* $M := \lceil t_f/h \rceil$.

In summary, for a given bounded reach set $\mathcal{R}_{t_f}(x_0)$ of a DTLHA $\mathcal{A}$ from x_0, the above results state the following: (1) A sampling period h can be determined for any given $\gamma \in \mathbb{R}^+$ so that the bounded reach set can be over-approximated. (2) If there is a discrete transition, then this event can be determined through the over-approximation of sampled states with appropriate values of δ and h. (3) If a discrete transition is deterministic and transversal, then an over-approximation of the discrete transition state can be computed with appropriate values of δ, γ, and h. (4) If every discrete transition state $x(\tau_k)$ is deterministic and transversal, then a bounded ϵ-reach set of $\mathcal{A}$ can be computed by appropriate values of δ, γ, and h.

3.2 Conditions for Determination of Deterministic and Transversal Discrete Transition

Now we elaborate in more detail on the result given in Lemma 11, especially on (iii) and (iv), to develop some conditions that are used in Chapter 4 when we discuss an architecture and the corresponding algorithm for a bounded ϵ-reach set computation of a DTLHA.

Lemma 12. *Given a location l_c, if $\mathcal{D}_{t-h}(\mathcal{B}_\delta(x_0)) \subset (Inv_c)^\circ$ and $\mathcal{D}_t(\mathcal{B}_\delta(x_0)) \subset Inv_c^C$ for some $\delta > 0$ and $h > 0$, where $\mathcal{B}_\delta(x_0)$ is a δ-neighborhood of the initial state x_0, then there is a discrete transition from the location l_c within time interval $(t-h, t)$.*

Proof. Note $\mathcal{D}_t(x_0) \in \mathcal{D}_t(\mathcal{B}_\delta(x_0))$, where $\mathcal{D}_t(x_0)$ is the reached state at time t from x_0. Similarly, $\mathcal{D}_{t-h}(x_0) \in \mathcal{D}_{t-h}(\mathcal{B}_\delta(x_0))$. From the hypothesis, $\mathcal{D}_t(x_0) \in Inv_c^C$ and $\mathcal{D}_{t-h}(x_0) \in (Inv_c)^\circ$. This implies that there exists $\tau \in (t-h, t)$ such that $\mathcal{D}_s(x_0) \in Inv_c^\circ$ for $s \in [t-h, \tau)$ and $\mathcal{D}_s(x_0) \in Inv_c^C$ for $s \in (\tau, t]$. Hence there is a discrete transition at some time $\tau \in (t-h, t)$. □

Lemma 13. *Given a polyhedron $\mathcal{P}_t$ at time t, suppose that there is a discrete transition from a location l_c to some other locations, i.e., $\mathcal{P}_{t-h} \subset (Inv_c)^\circ$ and $\mathcal{P}_t \subset Inv_c^C$ for some $h > 0$. Then the discrete transition is deterministic if there exists a location l_n such that $l_n \neq l_c$ and $\mathcal{P}_t \subset (Inv_n)^\circ$.*

Proof. By Definition 5, the result is trivially true. □

Lemma 14. *Given polyhedron $\mathcal{P}_t$ at time t, $\gamma > 0$, and $h > 0$ satisfying (3.2), suppose that there is a deterministic discrete transition from a location l_c to a location l_n, i.e., $\mathcal{P}_{t-h} \subset (Inv_c)^\circ$ and $\mathcal{P}_t \subset (Inv_n)^\circ$ for some $h > 0$. Then for any $\epsilon > 0$, the discrete transition is transversal if the following conditions hold.*

(i) $h < (dia(\mathcal{J}_{c,n})/2)/(2\bar{v})$,

(ii) $\mathcal{D}_0(\mathcal{J}_{c,n}, dia(\mathcal{J}_{c,n})/2) \subset (Inv_c \cup Inv_n)$, *and*

(iii) $\langle \dot{x}_c, \vec{n} \rangle \geq \epsilon \wedge \langle \dot{x}_n, \vec{n} \rangle \geq \epsilon, \quad \forall x \in \mathcal{V}(\mathcal{J}'_{c,n})$,

where $\mathcal{J}_{c,n} := \mathcal{D}_0(\mathcal{P}_t, \gamma) \cap Inv_c \cap Inv_n$, $\mathcal{J}'_{c,n} := \mathcal{D}_0(\mathcal{J}_{c,n}, dia(\mathcal{J}_{c,n})/2) \cap Inv_c \cap Inv_n$, $\bar{v}$ is as defined in (3.2), $\mathcal{V}(\mathcal{P})$ is a set of vertices of a polyhedron $\mathcal{P}$, $\vec{n}$ is an outward normal vector of ∂Inv_c, and $\dot{x}_i$ is the vector flow evaluated with respect to the LTI dynamics of location $l_i \in \mathbb{L}$.

Proof. First note that $\mathcal{P}_{t-h} \subset (Inv_c)^\circ$ and $\mathcal{P}_t \subset (Inv_n)^\circ$, since there is a deterministic discrete transition from l_c to l_n. Since γ and h satisfy (3.2), $\mathcal{P}_{t-h} \subset \mathcal{D}_0(\mathcal{P}_t, \gamma)$. In fact, $\cup_{z \in \mathcal{P}_{t-h}} x(\tau; z) \subset \mathcal{D}_0(\mathcal{P}_t, \gamma)$ for $\tau \in [0, h]$ where $x(\tau; z) := e^{A_c \tau} z + \int_0^\tau e^{A_c s} u_c ds$. Since $\mathcal{D}_{t-h}(x_0) \in \mathcal{P}_{t-h}$ and $\mathcal{D}_t(x_0) \in \mathcal{P}_t$, $\mathcal{D}_{\tau'}(x_0) \in \mathcal{J}_{c,n} := \mathcal{D}_0(\mathcal{P}_t, \gamma) \cap Inv_c \cap Inv_n$ for some $\tau' \in (t-h, t)$ where $\mathcal{D}_{\tau'}(x_0)$ is a discrete transition state from l_c to l_n at time τ'. Thus $\mathcal{J}_{c,n} \neq \emptyset$ (more precisely, $\mathcal{J}^\circ_{c,n} \neq \emptyset$) and it is in fact an over-approximation of the deterministic discrete transition state $x_{\tau'} \in Inv_c \cap Inv_n$.

Notice that if (i) holds, then $\|x(h; z) - z\| < dia(\mathcal{J}_{c,n})/4 < dia(\mathcal{J}_{c,n})/2$ for any $z \in \mathcal{J}_{c,n}$ since $\|x(h; z) - z\| \leq \bar{v} h$ where $x(h; z)$ is the state reached from z at time h under the LTI dynamics of the location l_n and $\bar{v}$ is as defined in (3.2). Also notice that if (ii) and (iii) hold, then for any $z' \in \mathcal{J}'_{c,n}$, z' satisfies the deterministic and transversal discrete transition condition in Definition 5. If we now consider the fact that $dia(\mathcal{J}'_{c,n}) \geq 2 \cdot dia(\mathcal{J}_{c,n})$, then $x(\tau; z) \in Inv^\circ_n$ for $\tau \in (0, h)$. Since $z \in \mathcal{J}_{c,n}$ is arbitrary, it is easy to see that $\mathcal{D}_\tau(\mathcal{J}_{c,n}) \in Inv^\circ_n$ for $\tau \in (0, h)$. □

3.3 Bounded ϵ-Reachability of a DTLHA with Finite Precision Calculations

The results in Sections 3.1 and 3.2 rely on the assumption that the following quantities can be computed exactly:

- $x(t; x_0) = e^{At} x_0 + \int_0^t e^{As} u ds$.
- $\mathcal{H} \cap \mathcal{P}$, where $\mathcal{H}$ is a hyperplane and $\mathcal{P}$ is a polyhedron.
- $hull(\mathcal{V})$, where $hull(\mathcal{V})$ is the convex hull of $\mathcal{V}$ that is a finite set of points in $\mathbb{R}^n$.

However, these exact computation assumptions cannot be satisfied in practice and we can only compute each of these with possibly arbitrarily small computation error. In this section, we extend the theory for bounded ϵ-reach set computation of a DTLHA presented in Sections 3.1 and 3.2 to incorporate the issue of numerical computation errors.

3.3.1 Approximate Numerical Computations

In the sequel, we use $a(x, y)$ to denote an approximate computation of x with $y \in \mathbb{R}^+$ as an upper bound on the approximation error. The precise definition depends on the type of x:

- If x is a vector or a matrix, then $\|x - a(x, y)\| \leq y$.
- If x is a set, then $d_H(x, a(x, y)) \leq y$ where $d_H(x, z)$ is the Hausdorff distance.

We assume that a set of subroutines or functions is available for approximately computing these quantities, which we use to compute a bounded ϵ-reach set. More precisely, for given μ_c and μ_h, $a(\mathcal{H} \cap \mathcal{P}, \mu_c)$ and $a(hull(\mathcal{V}), \mu_h)$ are available. Moreover, we also assume that a set of approximate computations, specifically $a(e^{At}, \sigma_e)$, $a(\int_0^t e^{A\tau} d\tau, \sigma_i)$, $a(A \cdot b, \sigma_p)$, and $a(u + v, \sigma_a)$, are available for computing $x(t; x_0)$ for given approximation errors $\sigma_e, \sigma_i, \sigma_p$, and σ_a. From these approximate computational capabilities, we can derive an upper bound on the approximation error, denoted as μ_x, for $x(t; x_0)$.

We first note that, for all approximate computations $a(x, y)$ that are used for computing $x(t; x_0)$, we have

$$(x - y \cdot \mathbf{1}_{n \times m}) \leq a(x, y) \leq (x + y \cdot \mathbf{1}_{n \times m}), \tag{3.4}$$

where $x \in \mathbb{R}^{n \times m}$ and $\mathbf{1}_{n \times m}$ is an n by m matrix whose every element is 1. With this, we derive μ_x as follows.

$$e^{At} - \sigma_e \cdot \mathbf{1}_{n \times n} \leq a(e^{At}, \sigma_e) \leq e^{At} + \sigma_e \cdot \mathbf{1}_{n \times n},$$

$$\begin{aligned} e^{At} x_0 - (\sigma_e |x_0| + \sigma_p) \cdot \mathbf{1}_{n \times 1} &\leq a(e^{At} x_0, \sigma_p) \\ &\leq e^{At} x_0 + (\sigma_e |x_0| + \sigma_p) \cdot \mathbf{1}_{n \times 1}. \end{aligned}$$

Similarly,

$$\int_0^t e^{As} ds - \sigma_i \cdot \mathbf{1}_{n \times n} \leq a(\int_0^t e^{As} ds, \sigma_i) \leq \int_0^t e^{As} ds + \sigma_i \cdot \mathbf{1}_{n \times n}.$$

$$\int_0^t e^{As}ds \cdot u - (\sigma_i|u| + \sigma_p) \cdot \mathbf{1}_{n\times 1} \leq a(\int_0^t e^{As}ds \cdot u, \sigma_p)$$
$$\leq \int_0^t e^{As}ds \cdot u + (\sigma_i|u| + \sigma_p) \cdot \mathbf{1}_{n\times 1}.$$

Hence, we have

$$x(t; x_0) - \delta_x \leq a(x(t; x_0), \delta_x) \leq x(t; x_0) + \delta_x,$$

where $\delta_x := (2\sigma_p + \sigma_a + \sigma_e|x_0| + \sigma_i|u|) \cdot \mathbf{1}_{n\times 1}$.

Now, we define μ_x as the maximum of $|\delta_x|$ over the continuous state space $\mathcal{X}$ and the control input domain $\mathcal{U}$,

$$\mu_x := \max_{x\in\mathcal{X}, u\in\mathcal{U}} |\delta_x|. \tag{3.5}$$

3.3.2 Incorporation of Finite Precision Calculations in Bounded ϵ-Reachability of a DTLHA

In this section, we extend the result given in Sections 3.1 and 3.2 to relax the infinite precision computation assumption. Especially, we extend the results in Lemmas 10, 12, 13, and 14.

We first discuss how the relation between h and γ in Lemma 10 can be changed under finite precision computation.

Lemma 15. *Let $\rho > 0$ be an upper bound on the approximation errors of $a(x(t), \rho)$ for some $x(t) \in \mathbb{R}^n$ and some time $t > 0$. Then for a given LTI system $\dot{x} = Ax + u$, if h satisfies $h < (\gamma - \rho)/(\|A\|\bar{x} + \|u\|)$ for any given $\gamma > \rho$, where $\bar{x}$ is as defined in (3.2), then the following property holds:*

$$\bigcup_{z\in\mathcal{B}_\rho(x(t))} x(\tau; z) \subset \mathcal{B}_\gamma(x(t)), \quad \forall \tau \in [0, h], \tag{3.6}$$

where $x(\tau; z) = e^{A\tau}z + \int_0^\tau e^{As}uds$ and $\mathcal{B}_y(x)$ is a y-neighborhood around x.

Proof. Notice that $a(x(t), \rho) \in \mathcal{B}_\rho(x(t))$, and for any $x(t) \in \mathcal{X}$,

$$\begin{aligned} \|x(t+h) - x(t)\| &\leq \int_t^{t+h} \|\dot{x}(s)\| ds \\ &\leq (\|A\|\bar{x} + \|u\|)h. \end{aligned}$$

Since $h < (\gamma - \rho)/(\|A\|\bar{x} + \|u\|)$, $\|x(t+h) - x(t)\| < \gamma - \rho$ for any $x(t) \in \mathcal{X}$. In fact, $\|x(t+s) - x(t)\| < \gamma - \rho$ for all $s \in [0, h]$. Hence for any $z \in \mathcal{B}_\rho(x(t))$, $x(s; z) \in \mathcal{B}_{\gamma-\rho}(z)$ for $s \in [0, h]$. This implies that for any $z \in \mathcal{B}_\rho(x(t))$, $\|x(t) - x(s; z)\| \leq \|x(t) - z\| + \|z - x(s; z)\| \leq \gamma$. □

Lemma 16. *Given $\rho > 0$ for $a(\mathcal{D}_t(\mathcal{B}_\delta(x_0)), \rho)$, let $\mathcal{P}_t := \mathcal{D}_t(\mathcal{B}_\delta(x_0), \rho)$. Then if h satisfies the inequality in (3.7) for a given $\gamma > \rho$, then $\mathcal{D}_\tau(\mathcal{P}_t) \subset \mathcal{D}_t(\mathcal{B}_\delta(x_0), \gamma)$ for all $\tau \in [0, h]$:*

$$h < \frac{\gamma - \rho}{\bar{v}}, \tag{3.7}$$

where $\bar{v}$ is as defined in (3.2).

Proof. Let $\mathcal{V}$ and $\mathcal{V}'$ be the set of extreme points of $\mathcal{D}_t(\mathcal{B}_\delta(x_0))$ and $\mathcal{P}_t$, respectively. Since (3.7) hold, we know from Lemma 15 that for each $x(t) \in \mathcal{V}$, $\mathcal{D}_\tau(\mathcal{B}_\rho(x(t))) \subset \mathcal{B}_\gamma(x(t))$ for all $\tau \in [0, h]$. Since for each $z \in \mathcal{V}'$, there exists $x(t) \in \mathcal{V}$ such that $\|x(t) - z\| \leq \rho$. Notice that for each $z \in \mathcal{V}'$, $z \in \mathcal{B}_\rho(x(t))$ for some $x(t) \in \mathcal{V}$. Therefore, $\mathcal{D}_\tau(\mathcal{P}_t) \subset \mathcal{D}_t(\mathcal{B}_\delta(x_0), \gamma)$ for all $\tau \in [0, h]$. □

In the sequel, for simplicity of notation, we use $\hat{x}$ to denote $a(x, \rho)$ for a given approximation error bound $\rho > 0$.

The condition (ii) in Lemma 11 enforces the size of an over-approximation of each sampled state along $\mathcal{R}_{t_f}(x_0)$ to be less than the given $\epsilon > 0$. Under the finite precision calculations, it is straightforward to extend the result of (ii) in Lemma 11, as shown in the following Lemma.

Lemma 17. *Given $\epsilon > 0$, $\rho > 0$, a polyhedron $\mathcal{P}$, and $\hat{\mathcal{P}}$, if $dia(\hat{\mathcal{P}}) < \epsilon - \rho$, then $dia(\mathcal{P}) < \epsilon$.*

Proof. Recall that $\hat{\mathcal{P}} := a(\mathcal{P}, \rho)$. This implies $d_H(\mathcal{P}, \hat{\mathcal{P}}) < \rho$. Hence $\mathcal{P} \subset \mathcal{D}_0(\hat{\mathcal{P}}, \rho)$. Notice that $dia(\mathcal{D}_0(\hat{\mathcal{P}}, \rho)) \leq dia(\hat{\mathcal{P}}) + \rho$. Hence $dia(\hat{\mathcal{P}}) < \epsilon - \rho$ implies $dia(\mathcal{P}) < \epsilon$. □

Now we address the issue of numerical computation errors in determining a deterministic and transversal discrete transition. The conditions developed in the following lemmas are sufficient in that if they are satisfied by a given polyhedron $\hat{\mathcal{P}}$ with a given approximation error ρ at time t, then there is a deterministic and transversal discrete transition at some time $\tau \in [t-h, t]$.

Lemma 18. *Given $\rho > 0$, a location l_c, and $\hat{\mathcal{D}}_t(\mathcal{B}_\delta(x_0))$ at time t, if*

(i) $\hat{\mathcal{D}}_{t-h}(\mathcal{B}_\delta(x_0),\, \rho) \subset (Inv_c)^\circ$, *and*

(ii) $\hat{\mathcal{D}}_t(\mathcal{B}_\delta(x_0), \rho) \subset Inv_c^C$

for some $\delta > 0$ and $h > 0$, then there is a discrete transition from the location l_c to some other locations.

Proof. Since $d_H(\mathcal{D}_t(\mathcal{B}_\delta(x_0)), \hat{\mathcal{D}}_t(\mathcal{B}_\delta(x_0))) \leq \rho$, $\mathcal{D}_t(\mathcal{B}_\delta(x_0)) \subset \hat{\mathcal{D}}_t(\mathcal{B}_\delta(x_0), \rho)$. Similarly, $\mathcal{D}_{t-h}(\mathcal{B}_\delta(x_0)) \subset \hat{\mathcal{D}}_{t-h}(\mathcal{B}_\delta(x_0), \rho)$. Hence $\mathcal{D}_t(\mathcal{B}_\delta(x_0)) \subset Inv_c^C$ and $\mathcal{D}_{t-h}(\mathcal{B}_\delta(x_0)) \subset (Inv_c)^\circ$. Then the result follows immediately from Lemma 12. □

Lemma 19. *Given $\rho > 0$, a location l_c, and a polyhedron $\mathcal{P}_t$ at time t, suppose that there is a discrete transition from a location l_c to some other locations, i.e., $\mathcal{D}_0(\hat{\mathcal{P}}_{t-h}, \rho) \subset (Inv_c)^\circ$ and $\mathcal{D}_0(\hat{\mathcal{P}}_t, \rho) \subset Inv_c^C$ for some $h > 0$. Then there is a deterministic discrete transition from l_c to l_n if there exists a location l_n such that $l_n \neq l_c$ and $\mathcal{D}_0(\hat{\mathcal{P}}_t, \rho) \subset (Inv_n)^\circ$.*

Proof. Since $\mathcal{P}_{t-h} \subset \mathcal{D}_0(\hat{\mathcal{P}}_{t-h}, \rho)$, $\mathcal{P}_{t-h} \subset (Inv_c)^\circ$. Similarly, $\mathcal{P}_t \subset (Inv_n)^\circ$ since $\mathcal{P}_t \subset \mathcal{D}_0(\hat{\mathcal{P}}_t, \rho)$. Then by Lemma 13, the conclusion holds. □

Lemma 20. *Given $\rho > 0$, $\gamma > 0$ and $h > 0$ satisfying (3.7), and a polyhedron $\mathcal{P}_t$ at time t, suppose that there is a deterministic discrete transition from a location l_c to a location l_n, i.e., $\mathcal{D}_0(\hat{\mathcal{P}}_{t-h}, \rho) \subset (Inv_c)^\circ$ and $\mathcal{D}_0(\hat{\mathcal{P}}_t, \rho) \subset (Inv_n)^\circ$ for some $h > 0$. Then for any $\epsilon > 0$, the discrete transition is transversal if the following conditions hold:*

(i) $h < (dia(\hat{\mathcal{J}}_{c,n})/2)/(2\bar{v})$,

(ii) $\mathcal{D}_0(\hat{\mathcal{J}}_{c,n}, dia(\hat{\mathcal{J}}_{c,n})/2 + \rho) \subset (Inv_c \cup Inv_n)$, *and*

(iii) $\langle \dot{x}_c, \vec{n} \rangle \geq \epsilon \wedge \langle \dot{x}_n, \vec{n} \rangle \geq \epsilon, \quad \forall x \in \mathcal{V}(\hat{\mathcal{J}}'_{c,n})$,

where $\hat{\mathcal{J}}_{c,n} := \mathcal{D}_0(\hat{\mathcal{P}}_t, \gamma+\rho) \cap Inv_c \cap Inv_n$, $\hat{\mathcal{J}}'_{c,n} := \mathcal{D}_0(\hat{\mathcal{J}}_{c,n},\, dia(\hat{\mathcal{J}}_{c,n})/2+\rho) \cap Inv_c \cap Inv_n$, *and* $\dot{x}_i$ *and* $\vec{n}$ *are as defined in Lemma 14.*

Proof. Notice that $\mathcal{D}_0(\mathcal{P}_t, \gamma) \subset \mathcal{D}_0(\hat{\mathcal{P}}_t, \gamma+\rho)$ since $d_H(\mathcal{P}_t,\, \hat{\mathcal{P}}_t) \leq \rho$. Then, by the definition of $\mathcal{J}_{c,n}$ given in Lemma 14 and $\hat{\mathcal{J}}_{c,n}$, we know $\mathcal{J}_{c,n} \subset \hat{\mathcal{J}}_{c,n}$. Hence, $\hat{\mathcal{J}}_{c,n} \neq \emptyset$, and in fact it is an over-approximation of the deterministic discrete transition state as is $\mathcal{J}_{c,n}$ in Lemma 14.

By the same argument used in the proof of Lemma 14, if (i) holds, then $\mathcal{D}_\tau(\hat{\mathcal{J}}_{c,n}) \subset \mathcal{D}_0(\hat{\mathcal{J}}_{c,n}, dia(\hat{\mathcal{J}}_{c,n})/2)$ for $\tau \in (0,h)$. Hence, (ii) and (iii) imply that $\mathcal{D}_\tau(\hat{\mathcal{J}}_{c,n}) \subset Inv_n^\circ$ for $\tau \in [0,h]$. Therefore, the conclusion holds since $\mathcal{J}_{c,n} \subset \hat{\mathcal{J}}_{c,n}$. □

CHAPTER 4

ARCHITECTURE AND ALGORITHM FOR COMPUTING A BOUNDED ϵ-REACH SET OF A DTLHA

In the algorithm and accompanying theoretical results for a bounded ϵ-reach set computation presented in Chapter 2, the issue of computation with finite precision has not been addressed. Moreover, even though the proposed algorithm can compute a bounded ϵ-reach set correctly, it is far from being computationally efficient since the algorithm restarts its ϵ-reach set computation from an initial state x_0 at time $t = 0$ whenever the values of δ and γ are changed. Moreover, the algorithm does not provide any flexibility in choosing the values for δ and γ whenever the algorithm needs to be continued with different δ and γ values, since only one specific decision rule, resulting in δ and γ which are monotonically decreasing, is tightly embedded within the algorithm. We address all these issues in this chapter.

We begin by presenting an architecture for a bounded ϵ-reach set computation followed by a new bounded ϵ-reach set algorithm that is based on the theoretical results developed in Chapter 3, so that the overall computation process can be better optimized in terms of computational efficiency and flexibility.

4.1 Architecture for Flexibility and Efficiency

One of the main objectives of the architectural design is to provide flexibility. We argue that this can be achieved by decoupling the part where decisions are made, called *Policy*, and the part where some specific steps of computation are performed, which is called *Algorithm* in our context (but called *Mechanism* in some other contexts). Figure 4.1 shows a proposed architecture based on this design principle. The proposed architecture consists of roughly four different parts which are Policy (*Policy* module), Algorithm (*Main Algorithm* and *Condition Checking* modules), Data (*System Description* and

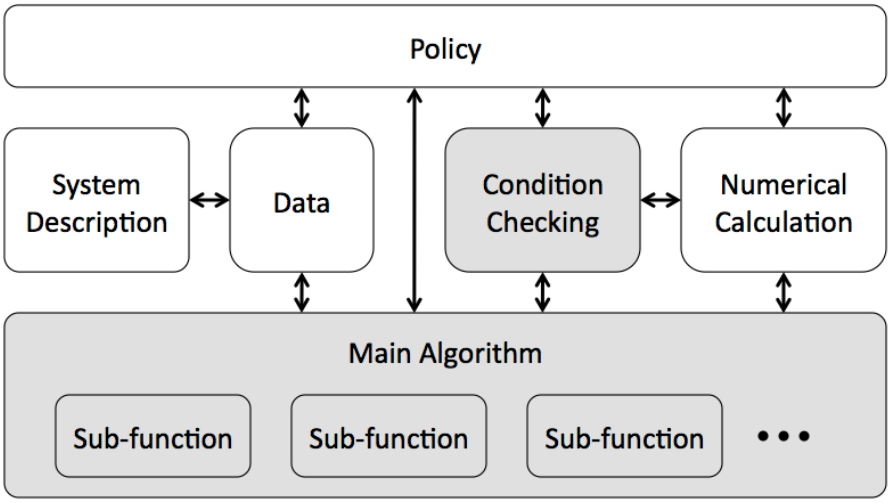


Figure 4.1: An architecture for bounded ϵ-reach set computation.

Data modules), and *Numerical Calculation* module. A more detailed explanation of each of these modules is given below.

Policy This module contains a user-defined rule to choose appropriate values of the parameters, especially δ and γ that are needed to continue to compute a bounded ϵ-reach set of a DTLHA when an ambiguous situation is encountered in the Main Algorithm module. Furthermore, this module can make a decision about the choice of numerical calculation algorithms, which affect the computational accuracy for each approximate numerical function defined in Section 3.3.1.

System Description The System Description module contains information about the system, described by a modeling language; it consists of $\mathcal{X}$, the domain of continuous state space, a DTLHA $\mathcal{A}$, and an initial continuous state x_0, an initial location l_0 (i.e., discrete state) where x_0 is contained. Also, to specify the required computation, an upper bound T on terminal time, an upper bound N on the total number of discrete transitions, and an approximation parameter ϵ, are described in the System Description module. In short, all information required to describe a problem of a bounded ϵ-reach set computation of a DTLHA is contained in the System Description module.

Data The data generated by the System Description module, called `SystemData`, is stored in the Data module which can then be used by the rest of the modules in the architecture. Furthermore, the data that are generated on-the-fly in a bounded ϵ-reach set computation by the Main Algorithm module, called `ReachSetHistory` and `TransitionHistory`, are also stored in this module.

Condition Checking To ensure a correct bounded ϵ-reach set computation, a bounded ϵ-reach set algorithm needs to correctly (i) detect a deterministic and transversal discrete transition if there is one, (ii) determine whether the size of the set computed as an over-approximation of the reach set between samples is smaller than the specified parameter ϵ, and (iii) check whether a sampling period h and an over-approximation parameter γ satisfy the relation for over-approximation guarantee. All functions which implement these condition checkings are contained in this module. More detail on this module is given in Section 4.2.

Main Algorithm With the inputs from the Policy module, the Main Algorithm computes a bounded ϵ-reach set utilizing Sub-functions and functions from the Condition Checking module until it either successfully finishes its computation, or cannot make further progress that happens when some required conditions are not met. If the algorithm encounters the latter situation, then it returns to the Policy module indicating the problems that the Policy module has to resolve so as to continue the computation. During a bounded ϵ-reach set computation, the Main Algorithm stores its computational state in two data structures, called `ReachSetHistory` and `TransitionHistory`, which are in turn stored in the Data module.

`ReachSetHistory` contains the computation results and the information used to produce the results at each step of computation, described in more detail in Section 4.2. One of the most important benefits of maintaining this information is that the computational efficiency can be improved significantly since the computation does not need to be restarted from the initial time and state whenever new parameter values, such as a smaller δ and γ, have to be used to continue the computation. Under this architecture, the Policy module can go back to any past computational step and make the Main Algorithm restart the computation from that point. In `TransitionHistory`,

the information about the discrete transition is stored. Maintaining this information in `TransitionHistory` also contributes to improving efficiency of the overall bounded ϵ-reach set computation process. More detail on this module is given in Section 4.2.

Numerical Calculation This module contains a collection of numerical functions for computing a matrix exponential, an integral of a matrix exponential, the intersection between polyhedra, a convex hull of a finite set of points, and so on. Each of these functions is in fact an implementation of some computational algorithms. As an example, $a(e^{At}, \sigma_e)$ can be computed in many different ways as shown in [22]. Each of the different algorithms can compute e^{At} with different accuracy. Hence, the computational accuracy of a bounded ϵ-reach set computation inevitably depends on the choice of the algorithms for computation of each of the $a(x, y)$'s assumed above. We decouple such issues arising in the low-level numerical calculations from our proposed bounded ϵ-reach set algorithm, which is the reason for the separate module for numerical calculation in our architecture.

4.2 Algorithm for Bounded ϵ-Reach Set of a DTLHA

The proposed algorithm for a bounded ϵ-reach set computation is decomposed into roughly two parts, the Main Algorithm module and the Condition Checking module. In this section, we discuss these modules in more detail. Recall that we use $\hat{x}$ to denote $a(x, \rho)$ for some given approximation error bound $\rho \in \mathbb{R}^+$. In particular, a polyhedron $\hat{\mathcal{P}}$ in the sequel should be understood as an approximation of a polyhedron $\mathcal{P}$, i.e., $\hat{\mathcal{P}} := a(\mathcal{P}, \rho)$.

4.2.1 Condition Checking Module

In computing a bounded ϵ-reach set, the following set of questions needs to be answered at each step of computation in the Main Algorithm to produce a correct result:

1. Given δ and γ, is the diameter of $\mathcal{D}_t(\mathcal{B}_\delta(x_0), \gamma)$ at current sample time t less than the given ϵ?

2. Given δ, γ, and h, is $\mathcal{D}_\tau(\mathcal{B}_\delta(x_0)) \subset \mathcal{D}_t(\mathcal{B}_\delta(x_0), \gamma)$ for all $\tau \in [t, t+h]$ at current sample time t?

3. Given h, δ, and $\mathcal{D}_t(\mathcal{B}_\delta(x_0))$, can we conclude that a discrete transition has occurred within the time interval $t-h$ and t?

4. If there is a discrete transition as above, is it a deterministic discrete transition?

5. If there is a deterministic discrete transition above, is it a transversal discrete transition?

Corresponding to these questions, the Condition Checking module consists of the following set of functions which are based on the results in Section 3.3.2.

IsEpsilonSmall$(\cdot)$ Given $\epsilon > 0$ and a polyhedron $\hat{\mathcal{P}}$, this function determines whether $dia(\mathcal{P}) < \epsilon$ or not, where $dia(\mathcal{P})$ denotes the diameter of a polyhedron $\mathcal{P}$. As shown in Lemma 17, $dia(\mathcal{P}) < \epsilon$ if $dia(\hat{\mathcal{P}}) < \epsilon - \rho$. Hence, this function returns `true` if $dia(\hat{\mathcal{P}}) < \epsilon - \rho$.

IsOverApproximate$(\cdot)$ Given γ and ρ, this function determines whether a sampling period h and γ satisfy the condition (3.7). Hence, if $h < (\gamma - \rho)/\bar{v}$, this function returns `true` where $\bar{v}$ is as defined in (3.2).

IsTransition$(\cdot)$ Given a sampling period h, a location l_c, and a polyhedron $\hat{\mathcal{P}}_t$ at time t, this function checks if there is a discrete transition from a location l_c at some time in between $t-h$ and t. In Lemma 18, it is shown that if $\mathcal{D}_0(\hat{\mathcal{P}}_{t-h}, \rho) \subset (Inv_c)^\circ$ and $\mathcal{D}_0(\hat{\mathcal{P}}_t, \rho) \subset Inv_c^C$, then there is indeed a discrete transition at some time in $(t-h, t)$. Assuming that $\mathcal{D}_0(\hat{\mathcal{P}}_{t-h}, \rho) \subset (Inv_c)^\circ$ is satisfied at time $t-h$, this function returns `true` if $\mathcal{D}_0(\hat{\mathcal{P}}_t, \rho) \subset Inv_c^C$. If $\mathcal{D}_0(\hat{\mathcal{P}}_t, \rho) \subset (Inv_c)^\circ$, then this function returns `false`. In the other cases that $(\mathcal{D}_0(\hat{\mathcal{P}}_t, \rho) \cap Inv_c \neq \emptyset) \wedge (\mathcal{D}_0(\hat{\mathcal{P}}_t, \rho) \cap Inv_c^C \neq \emptyset)$, this function returns `error` to inform that other values of δ or h need to be used to resolve the ambiguity.

IsDeterministic$(\cdot)$ Given a location l_c and a polyhedron $\hat{\mathcal{P}}$, this function checks if a discrete transition from l_c is a deterministic transition to some other location l_n. Based on the result in Lemma 19, this function returns

the location l_n if there is a location $l_n \in \mathbb{L}$ such that $l_n \neq l_c$ and $\mathcal{D}_0(\hat{\mathcal{P}}, \rho) \subset (Inv_n)^\circ$. Otherwise it returns `error`.

IsTransversal$(\cdot)$ Given h, γ, and a polyhedron $\hat{\mathcal{P}}$, this function checks if a discrete transition from a location l_c to other location l_n is a transversal discrete transition or not, using the conditions (i), (ii), and (iii) in Lemma 20. If it is a transversal discrete transition, this function returns $\hat{\mathcal{D}}_h(\hat{\mathcal{J}}_{c,n}, \rho')$, where $\hat{\mathcal{D}}_h(\hat{\mathcal{J}}_{c,n})$ is an approximation of $\mathcal{D}_h(\hat{\mathcal{J}}_{c,n})$ which is the image of $\hat{\mathcal{J}}_{c,n}$ at time h under the linear dynamics of a location l_n. Otherwise, it returns `error`. Note that ρ' is a numerical calculation error that is introduced during the computation of $\mathcal{D}_h(\hat{\mathcal{J}}_{c,n})$ from $\hat{\mathcal{J}}_{c,n}$.

4.2.2 Main Algorithm Module

Roughly, the Main Algorithm module consists of two parts. The first part is a function called `ReachSet`$(\cdot)$ which is the main function to compute a bounded ϵ-reach set, and the second part is a set of functions called *Sub-functions* which are called by `ReachSet`$(\cdot)$ during its computation. We first describe the functions defined as Sub-functions.

ReachNext$(\cdot)$ Given h, γ, and a polyhedron $\hat{\mathcal{P}}$, this function returns $\hat{\mathcal{D}}_h(\hat{\mathcal{P}})$, an approximation of the linear image of a polyhedron $\hat{\mathcal{P}}$ at time h under a linear dynamics, and $\hat{\mathcal{D}}_h(\hat{\mathcal{P}}, \gamma)$, an over-approximation of $\hat{\mathcal{D}}_h(\hat{\mathcal{P}})$ for a given over-approximation parameter γ. This function also returns estimates of the upper bound of computation errors ρ' and ρ'' along with $\hat{\mathcal{D}}_h(\hat{\mathcal{P}})$ and $\hat{\mathcal{D}}_h(\hat{\mathcal{P}}, \gamma)$, so that `ReachSet(·)` function can keep track of the numerical errors accumulated from the initial time up to the current time t. Notice that ρ' and ρ'' are defined via $d_H(\mathcal{D}_h(\mathcal{P}), \hat{\mathcal{D}}_h(\hat{\mathcal{P}})) \leq \rho'$ and $d_H(\mathcal{D}_h(\mathcal{P}, \gamma), \hat{\mathcal{D}}_h(\hat{\mathcal{P}}, \gamma)) \leq \rho''$, respectively.

To compute $\hat{\mathcal{D}}_h(\hat{\mathcal{P}})$ and $\hat{\mathcal{D}}_h(\hat{\mathcal{P}}, \gamma)$, this function exploits the fact that the polyhedral structure is preserved under a linear dynamics in the following way. Given polyhedron $\hat{\mathcal{P}}$, this function first computes $\mathcal{V}(\hat{\mathcal{P}})$ which is a set that contains the vertices of $\hat{\mathcal{P}}$, and possibly some other points in $\hat{\mathcal{P}}$. [1]

[1]The reason for allowing some other points that are possibly not vertices is because $\hat{\mathcal{P}}$ is itself computed as the linear image of a finite number of points, and we would like to avoid the need to computationally determine precisely which remain extreme points under

Then for each $v_i \in \mathcal{V}(\hat{\mathcal{P}})$, it computes $v_i(h) := e^{Ah}v_i + \int_0^h e^{As}uds$ where A and u are given by the linear dynamics of a location on which the linear image of $\hat{\mathcal{P}}$ is computed. If we let $\mathcal{V}_h(\hat{\mathcal{P}}) := \{v_i(h) : v_i \in \mathcal{V}(\hat{\mathcal{P}})\}$, then $\mathcal{D}_h(\hat{\mathcal{P}}) := hull(\mathcal{V}_h(\hat{\mathcal{P}}))$ where $hull(\mathcal{V}_h(\hat{\mathcal{P}}))$ is the convex hull of $\mathcal{V}_h(\hat{\mathcal{P}})$. Notice that what we really have here is $\hat{\mathcal{D}}_h(\hat{\mathcal{P}})$, and not $\mathcal{D}_h(\hat{\mathcal{P}})$, since there is a numerical calculation error in the $hull(\mathcal{V}_h(\hat{\mathcal{P}}))$ computation. From $\mathcal{V}_h(\hat{\mathcal{P}})$, this function can also compute $\hat{\mathcal{D}}_h(\hat{\mathcal{P}}, \gamma)$ easily. For each $v_i(h) \in \mathcal{V}_h(\hat{\mathcal{P}})$, it first constructs a hypercubic γ-neighborhood of $v_i(h)$. If we denote such a neighborhood by $\mathcal{B}_\gamma(v_i(h))$, then the convex hull of the set of vertices of $\mathcal{B}_\gamma(v_i(h))$ for all $v_i(h) \in \mathcal{V}_h(\hat{\mathcal{P}})$ defines a $\hat{\mathcal{D}}_h(\hat{\mathcal{P}}, \gamma)$.

AtTransition$(\cdot)$ This function is called by `ReachSet(·)` when a discrete transition from a given location l_c is detected by `IsTransition(·)`. Then this function internally calls `IsDeterministic(·)` and `IsTransversal(·)` functions to check if this discrete transition is deterministic and transversal. If it is, then this function returns a location l_n that is returned by `IsDeterministic(·)` and $\hat{\mathcal{D}}_h(\hat{\mathcal{J}}_{c,n}, \rho')$ that is returned by `IsTransversal(·)`. However, if any of these functions returns `error`, this function returns the same `error` to indicate the necessity of a decision in the Policy module to resolve the erroneous situation.

ImageAt$(\cdot)$ Even though the overall computational efficiency of a bounded ϵ-reach set computation can be improved by the proposed architecture, it is unavoidable to restart the computation from an initial state when the value of parameter δ which defines an initial neighborhood around an initial state is changed. If the algorithm encounters such a situation, `ImageAt`$(\cdot)$ can be used to reduce the number of computational steps. Given t and δ, the goal of this function is to compute $\mathcal{D}_t(\mathcal{B}_\delta(x_0))$. More precisely, this function computes $\hat{\mathcal{D}}_t(\mathcal{B}_\delta(x_0))$ and a corresponding numerical calculation error ρ such that $d_H(\mathcal{D}_t(\mathcal{B}_\delta(x_0)), \hat{\mathcal{D}}_t(\mathcal{B}_\delta(x_0))) \leq \rho$. To compute $\hat{\mathcal{D}}_t(\mathcal{B}_\delta(x_0))$ from $\mathcal{B}_\delta(x_0)$, what this function needs to know is the computational history of `ReachSet(·)` containing the time τ_k when a discrete transition is detected, and the values of the parameters h and γ that were used at the time τ_k. Note that all of this information is stored in `TransitionHistory` by `ReachSet(·)`.

---

the linear map.

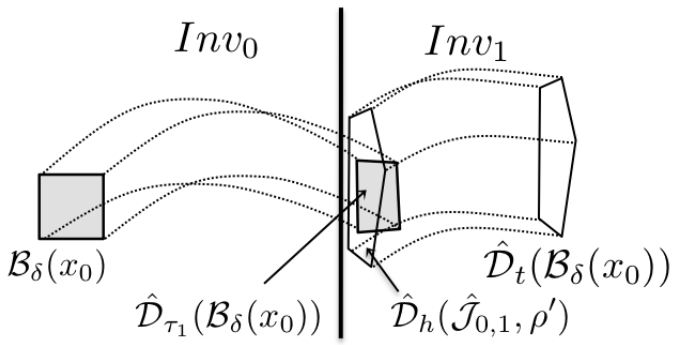


Figure 4.2: Illustration of the computation in `ImageAt`$(\cdot)$.

To show how $\hat{\mathcal{D}}_t(\mathcal{B}_\delta(x_0))$ is computed, we consider an example shown in Figure 4.2 that computes $\hat{\mathcal{D}}_t(\mathcal{B}_\delta(x_0))$ in a location l_1 from $\mathcal{B}_\delta(x_0)$ in a location l_0. If we know τ_1, then $\hat{\mathcal{D}}_{\tau_1}(\mathcal{B}_\delta(x_0))$ and its corresponding numerical calculation error can easily be computed as explained in `ReachNext(·)`. Now, to compute $\hat{\mathcal{D}}_h(\hat{\mathcal{J}}_{0,1}, \rho')$, this function calls `AtTransition(·)`. Note that the location l_1 is determined by `IsDeterministic(·)` and $\hat{\mathcal{D}}_h(\hat{\mathcal{J}}_{0,1}, \rho')$ is returned by the `IsTransversal(·)` function from $\hat{\mathcal{D}}_{\tau_1}(\mathcal{B}_\delta(x_0))$. If `AtTransition(·)` returns `error`, then `ImageAt(·)` returns the same `error`. Otherwise, this function continues its image computation to compute $\hat{\mathcal{D}}_t(\mathcal{B}_\delta(x_0))$ from $\hat{\mathcal{D}}_h(\hat{\mathcal{J}}_{0,1}, \rho')$ under the linear dynamics of a location l_1.

Now, we describe the main function, called `ReachSet`$(\cdot)$, in the Main Algorithm.

ReachSet$(\cdot)$ Given an input (k, δ, γ, h) from the Policy module, where k indicates one of the past computation steps of `ReachSet(·)` from which this function starts its computation, this function computes a bounded ϵ-reach set by utilizing all other functions in the Main Algorithm and the Condition Checking modules. This function first retrieves the computation data at the $(k-1)$-th computation step from the `ReachSetHistory` and starts its k-th computation step using this data. As shown in Algorithm 3, it continues its computation until it either successfully computes a bounded ϵ-reach set or encounters some `error`. If there is an `error` from any of the functions that are called, then this function returns the same `error` to the Policy module to indicate the cause of the `error`. For each type of `error`, `ReachSet`$(\cdot)$ expects to have a new input from the Policy module to continue

Algorithm 3: Algorithm of `ReachSet`$(\cdot)$.

Input: $k, \delta_k, \gamma_k, h_k, \sigma_e, \sigma_i, \sigma_p, \sigma_a, \mu_c, \mu_h$
Result: ReachSetHistory, TransitionHistory

`compute` μ_x from $(\sigma_e, \sigma_i, \sigma_p, \sigma_a)$
while true **do**
  `Get` $(k-1)$-th computation data from ReachSetHistory
  if $\delta_k \neq \delta_{k-1}$ **then**
    `call ImageAt`$() \to \hat{\mathcal{D}}_{t_{k-1}}(\mathcal{B}_{\delta_k}(x_0))$
    if error **then return** error
  end
  if `IsOverApproximate`$() =$ false **then return** error
  $t_k = t_{k-1} + h_k$
  `call ReachNext`$() \to \{\hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0)), \hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0), \gamma_k)\}$
  `compute` ρ_k s.t. $d_H(\mathcal{D}_{t_k}(\mathcal{B}_{\delta_k}(x_0), \gamma_k), \hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0), \gamma_k)) \leq \rho_k$
  if `IsEpsilonSmall`$() =$ false **then return** error
  `call IsTransition`$() \to$ out
  if out $=$ error **then return** error
  else if out $=$ false **then** $l_k \leftarrow l_{k-1}$
  else if out $=$ true **then**
    `call AtTransition`$() \to \{l_k, \hat{\mathcal{P}}\}$
    if error **then return** error
    $\hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0)) \leftarrow \hat{\mathcal{P}}$
    $\hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0), \gamma_k) \leftarrow \hat{\mathcal{D}}_0(\hat{\mathcal{P}}, \gamma_k)$
    `update` ρ_k
    jump $\leftarrow$ jump $+ 1$
    `store` $\{t_k, l_k, h_k\}$ to TransitionHistory
  end
  `store` to ReachSetHistory the data of
    $\{k, t_k, l_k, \delta_k, \gamma_k, h_k, \rho_k, \hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0)), \hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0), \gamma_k)\}$
  $k \leftarrow k + 1$
  if $(t_k \geq T) \vee$ (jump $\geq N$) **then return** done
end

its computation. Besides the input (k, δ, γ, h), an additional set of inputs $(\sigma_e, \sigma_i, \sigma_p, \sigma_a, \mu_c, \mu_h)$ can also be provided by the Policy module when there are numerical computational algorithms with better computational accuracies in the Numerical Calculation modules to resolve an erroneous situation occurring in `ReachSet(·)`.

As mentioned in Section 4.1, `ReachSet(·)` stores its computation results (or states) in `ReachSetHistory` data structure at every step of its computation. The information stored in `ReachSetHistory` includes k that is the step of its computation, and the time t_k at the k-th computation step, $(\delta_k, \gamma_k, h_k)$ that are used in the k-th computation step without causing any `error`, and $\hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0))$ and $\hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0), \gamma_k)$ along with their corresponding numerical computation errors, ρ'_k and ρ''_k. In addition to `ReachSetHistory`, `ReachSet(·)` maintains another data structure, called `TransitionHistory` which contains computation information of `ReachSet(·)` only at the time of discrete transition between locations, intended to be used in `ImageAt(·)`.

We now have the following overall main result:

Theorem 4. *For a given* `SystemData` $:= (\mathcal{X}, \mathcal{A}, l_0, x_0, T, N, \epsilon)$*, if* `ReachSet`$(\cdot)$ *in Algorithm 3 returns* `done`*, then a bounded ϵ-reach set of a DTLHA $\mathcal{A}$ over the continuous domain $\mathcal{X}$ from an initial state $x_0 \in Inv_0$, denoted as $\mathcal{R}_{t_f}(x_0, \epsilon)$, is the following:*

$$\mathcal{R}_{t_f}(x_0, \epsilon) := \bigcup_{k=1}^{K} \hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0), \gamma_k), \tag{4.1}$$

where K is the number of data elements in `ReachSetHistory`*, $t_f := \min\{T, \tau_N\}$, and τ_N is the N-th discrete transition.*

Proof. For each $k \leq K$, (i) (γ_k, h_k) satisfies Lemma 16, and (ii) $\hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0), \gamma_k)$ satisfies Lemma 17. These imply that $\hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0), \gamma_k)$ is guaranteed to be a correct γ_k-approximation of $\mathcal{D}_{t_k}(\mathcal{B}_{\delta_k}(x_0))$ by γ_k and h_k, i.e.,

$$\bigcup_{\tau \in [0, h_k]} \mathcal{D}_{t_k+\tau}(\mathcal{B}_{\delta_k}(x_0)) \subset \hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0), \gamma_k). \tag{4.2}$$

Moreover $dia(\hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0)\ , \gamma_k)) < \epsilon - \rho_k$. If a deterministic and transversal discrete transition is detected at the k-th step by $\hat{\mathcal{D}}_{t_k}(\mathcal{B}_{\delta_k}(x_0))$, then (iii) by Lemmas 18, 19, and 20, there is in fact a deterministic and transversal dis-

crete transition in (t_{k-1}, t_k). This implies that a deterministic and transversal discrete transition event is correctly determined by `ReachSet(·)`. Finally, the fact that `done` is returned by `ReachSet`$(\cdot)$ implies that either $t_k > T$ or `jump` $> N$. Hence, t_f is $\min\{T, \tau_N\}$. Therefore, we conclude that $\mathcal{R}_{t_f}$ is a bounded ϵ-reach set of $\mathcal{A}$ from x_0. □

CHAPTER 5

OPTIMIZATION AND IMPLEMENTATION OF THE BOUNDED ϵ-REACH SET ALGORITHM

A prototype implementation of the proposed algorithm for a bounded ϵ-reach set of a DTLHA has been developed on Matlab. We use the Multi-Parametric Toolbox [23] for polyhedral operations. For other types of operations to compute $x(t; x_0) = e^{At}x_0 + \int_0^t e^{As}uds$, we use the built-in Matlab matrix arithmetic operations.

5.1 Implementation of the Algorithm

We begin this chapter by presenting some techniques that can improve the capabilities of the proposed algorithm in terms of the overall computational efficiency and detection of a discrete transition. More precisely, we introduce (i) a state dependent rule in choosing a sampling period h and (ii) a technique to compute a tight over-approximation of a discrete transition state.

For simplicity of the discussion, the issue of finite precision computation is not considered in the sequel.

5.1.1 State Dependent Choice of Sampling Period h

As shown in Lemma 10, a sampling period h needs to satisfy the following inequality condition to yield a given over-approximation with parameter γ:

$$h < \frac{\gamma}{\bar{v}}, \tag{5.1}$$

where $\bar{v} := \max_{l_i \in \mathbb{L}}\{\|A_i\|\bar{x} + \|u_i\|\}$ and $\bar{x} := \max_{x \in \mathcal{X}} \|x\|$.

Notice that the upper bound on h imposed in this inequality is constant and is conservative. If we have a rule for choosing a sampling period h in a less conservative manner by avoiding the use of $\bar{v}$ in (5.1), then it can

both (i) reduce the overall number of computation steps and (ii) increase the chance of detecting a discrete transition event, since the larger the sampling period h, the longer the distance between two consecutive sets $\mathcal{D}_t(\mathcal{B}_\delta(x_0))$ and $\mathcal{D}_{t+h}(\mathcal{B}_\delta(x_0))$. Hence, in this section, we derive such a rule for h that can determine h based on the information about $\mathcal{D}_t(\mathcal{B}_\delta(x_0))$ available at the current time t.

More formally, we derive a state dependent rule for sampling period h satisfying all the following three conditions for given ϵ and γ:

(I) $dia(\mathcal{D}_0(\mathcal{P}_t, \gamma)) \leq \epsilon$,

(II) $dia(\mathcal{D}_h(\mathcal{P}_t, \gamma)) \leq \epsilon$, and

(III) $\mathcal{D}_\tau(\mathcal{P}_t) \subset \mathcal{D}_0(\mathcal{P}_t, \gamma)$ for all $\tau \in [0, h]$,

where $\mathcal{P}_t$ is a set of reached states at time t, from an initial δ-neighborhood around x_0, i.e., $\mathcal{B}_\delta(x_0)$.

Proposition 1. *Given $\epsilon > 0$ and a polyhodron $\mathcal{P}_t$, a sampling period h determined by the following equation satisfies (I), (II), and (III) above.*

$$h = \frac{\epsilon - dia(\mathcal{P}_t)}{4(\|A\|\bar{y} + \|u\|)}, \tag{5.2}$$

where $\bar{y} := \max_{y \in \mathcal{P}_t} \|y\|$.

Proof. For given ϵ, γ, and $\mathcal{P}_t$, we first note $dia(\mathcal{D}_0(\mathcal{P}_t, \gamma)) \leq dia(\mathcal{P}_t) + 2\gamma$. Hence, it is easy to see that (I) is satisfied if the following holds:

$$dia(\mathcal{P}_t) + 2\gamma \leq \epsilon. \tag{5.3}$$

Similarly, we have $dia(\mathcal{D}_h(\mathcal{P}_t, \gamma)) \leq dia(\mathcal{D}_h(\mathcal{P}_t)) + 2\gamma$ for (II). To eliminate $\mathcal{D}_h(\mathcal{P}_t)$ from this relation, we use

$$\|x(h; y) - x(h; z)\| \leq e^{\|A\|h}\|y - z\|, \tag{5.4}$$

where $x(h; y) := e^{Ah}y + \int_0^h e^{As}uds$ and $x(h; z) := e^{Ah}z + \int_0^h e^{As}uds$ for some $A \in \mathbb{R}^{n \times n}$ and $u \in \mathbb{R}^{n \times 1}$. Then by the definition of $dia(\mathcal{P}) := \max_{y,z \in \mathcal{V}(\mathcal{P})} \|y - z\|$ for a polyhedron $\mathcal{P}$, we can rewrite (5.4) as follows.

$$dia(\mathcal{D}_h(\mathcal{P}_t)) \leq e^{\|A\|h} dia(\mathcal{P}_t). \tag{5.5}$$

From (5.5), we now have a sufficient condition for (II), without $\mathcal{D}_h(\mathcal{P}_t)$, as follows:

$$e^{\|A\|h} dia(\mathcal{P}_t) + 2\gamma \leq \epsilon. \tag{5.6}$$

We now derive a sufficient condition for (III). Here, the condition that h needs to satisfy is the following.

$$\max_{\tau \in [0,h]} \|x(\tau; y) - y\| < \gamma \qquad \forall y \in \mathcal{P}_t. \tag{5.7}$$

If we consider $x(s; y) := e^{As} y + \int_0^s e^{Av} u dv$ for $s \in [0, \tau]$, then we have

$$\begin{aligned} \|x(s; y)\| &\leq e^{\|A\|s} \|y\| + \int_0^s e^{\|A\|v} \|u\| dv \\ &= e^{\|A\|s} \|y\| + \frac{1}{\|A\|} \left(e^{\|A\|s} - 1 \right) \|u\|. \end{aligned} \tag{5.8}$$

Hence

$$\|x(s; y)\| \leq \max_{s \in [0,\tau]} \|x(s; y)\| \leq e^{\|A\|\tau} \|y\| + \frac{1}{\|A\|} \left(e^{\|A\|\tau} - 1 \right) \|u\|. \tag{5.9}$$

Then if we integrate the left and right hand sides of (5.9) over $s \in [0, \tau]$,

$$\int_0^\tau \|x(s; y)\| ds \leq \left\{ \|y\| e^{\|A\|\tau} + \frac{\|u\|}{\|A\|} \left(e^{\|A\|\tau} - 1 \right) \right\} \tau. \tag{5.10}$$

Then from (5.10), we can derive an upper bound of $\|x(\tau; y) - y\|$ as follows.

$$\begin{aligned} \|x(\tau; y) - y\| &\leq \int_0^\tau \|\dot{x}(s; y)\| ds \\ &\leq \int_0^\tau \{\| A\| \|x(s, y)\| + \|u\|\} ds \\ &= \|A\| \int_0^\tau \|x(s, y)\| ds + \|u\| \tau \\ &\leq \|A\| \left\{ \|y\| e^{\|A\|\tau} + \frac{\|u\|}{\|A\|} \left(e^{\|A\|\tau} - 1 \right) \right\} \tau + \|u\| \tau \\ &\leq \left(\|A\| \bar{y} + \|u\| \right) e^{\|A\|\tau} \tau, \end{aligned} \tag{5.11}$$

where $\bar{y} := \max_{y \in \mathcal{P}_t} \|y\|$.

Hence, (5.7) holds if the following holds:

$$\max_{\tau\in[0,h]} \left(\|A\|\bar{y} + \|u\|\right) e^{\|A\|\tau}\tau = \left(\|A\|\bar{y} + \|u\|\right) e^{\|A\|h}h < \gamma. \tag{5.12}$$

To simplify further, we now suppose that (5.3) and (5.6) both hold. Then we can eliminate $e^{\|A\|h}$ term from (5.12) as follows:

$$\left(\|A\|\bar{y} + \|u\|\right) e^{\|A\|h}h \leq \left(\|A\|\bar{y} + \|u\|\right) \frac{\epsilon - 2\gamma}{dia(\mathcal{P}_t)}h < \gamma. \tag{5.13}$$

Then we have

$$h < \frac{\gamma}{\epsilon - 2\gamma} \frac{dia(\mathcal{P}_t)}{\left(\|A\|\bar{y} + \|u\|\right)}. \tag{5.14}$$

If we choose h as follows, and consider $\gamma := (\epsilon - dia(\mathcal{P}_t))/2$, then we finally have

$$h := \frac{\gamma/2}{\epsilon - 2\gamma} \frac{dia(\mathcal{P}_t)}{\left(\|A\|\bar{y} + \|u\|\right)} = \frac{\epsilon - dia(\mathcal{P}_t)}{4(\|A\|\bar{y} + \|u\|)}. \tag{5.15}$$

□

To compare the sampling periods computed from (5.2) and (5.1), an example is considered of an LTI system $\dot{x} = Ax + u$ over a continuous state space $\mathcal{X} := [-10, 10] \times [-10, 10] \subset \mathbb{R}^2$, where A and u are defined as follows:

$$A = \begin{pmatrix} -0.2 & -1 \\ 3 & -0.2 \end{pmatrix} \qquad u = \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}.$$

Figure 5.1 shows the difference between the sampling period $h(x)$ determined by (5.2) and the sampling period h detemined by (5.1), where $\gamma := (\epsilon - dia(\mathcal{P}_t))/2$ is used as in (5.2). From (5.1), a constant value 0.0029 sec. is obtained for a sampling period h. As shown in Figure 5.1, (5.2) provides larger values of sampling period than the one from (5.1). More precisely, the maximum difference of $h(x) - h$ is 0.0441 sec., and the minimum difference is 0.

5.1.2 Tight Over-approximation of Discrete Transition State

As explained in Section 4.2, especially with respect to the `IsTransversal`$(\cdot)$ function, the radius of the size of the over-approximated reach set right after a discrete transition, i.e., $\mathcal{D}_t(\mathcal{B}_\delta(x_0))$ for some time t, increases roughly by the

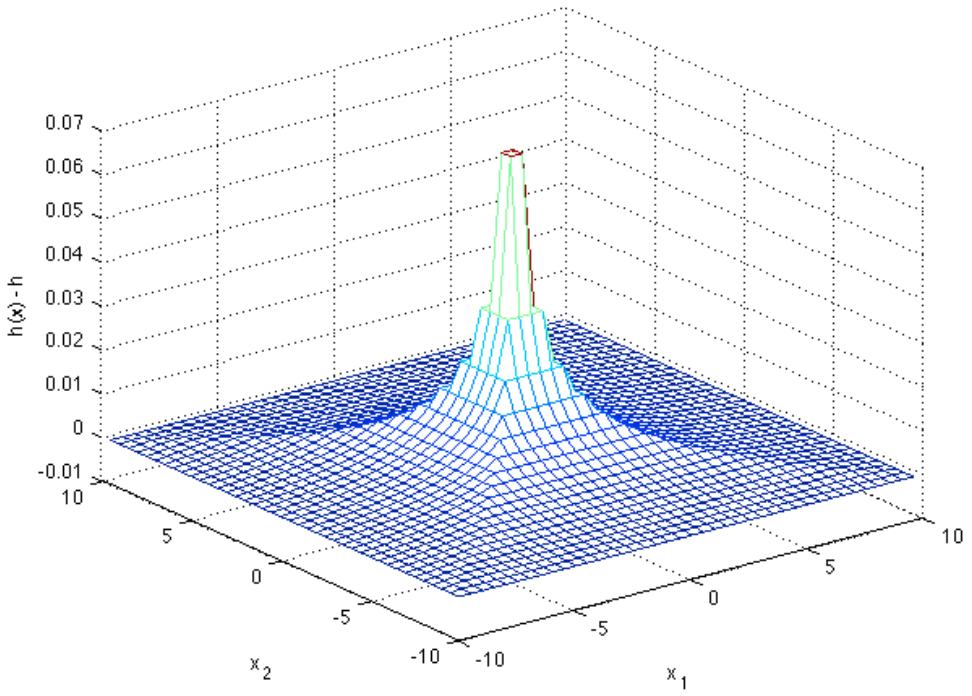


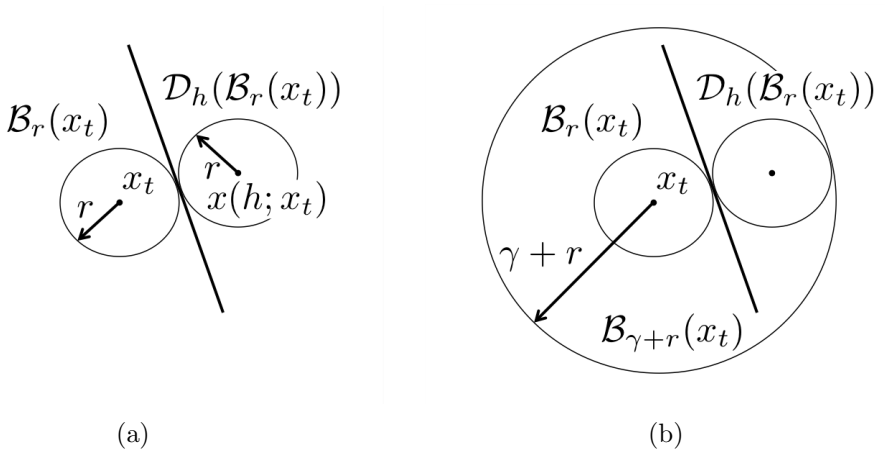
Figure 5.1: Sampling period h determined by (5.2).

amount γ that is used at time t through the computation of $\hat{\mathcal{J}}_{c,n}$. However, this can affect the capability of determination of a discrete transition event in the proposed algorithm since, as shown in (5.1) and (5.3), the maximum value for γ and correspondingly h becomes less as the diameter of $\mathcal{D}_t(\mathcal{B}_\delta(x_0))$ becomes larger, for given ϵ and $\mathcal{D}_t(\mathcal{B}_\delta(x_0))$.

In this section, we first analyze how the γ increment at every discrete transition affects the number of discrete transitions that can be effectively computed through the proposed algorithm. Subsequent to this, we propose a technique that can be used to overcome this issue. For simplicity of the analysis, in the sequel, we assume that the linear map is neither contractive nor expansive.

An Upper Bound on the Number of Discrete Transitions

Consider a situation at time t shown in Figure 5.2(a). Then for given $r \in \mathbb{R}^+$ and a polyhedron $\mathcal{B}_r(x_t)$ at time t, a sampling period h should satisfy the following to determine an event of discrete transition at this situation:



(a) (b)

Figure 5.2: Determination of a discrete transition via over-approximations of x_t and $x(h; x_t)$ at some time t.

$$\|x(h;y) - y\| > 2r \qquad \forall y \in \mathcal{B}_r(x_t). \tag{5.16}$$

If we let $\underline{v} := \min_{y \in \mathcal{B}_r(x_t)} \|\dot{y}\|$, then for all $y \in \mathcal{B}_r(x_t)$, we have

$$\|x(h;y) - y\| \geq \int_0^h \underline{v} ds = \underline{v} h > 2r. \tag{5.17}$$

If we assume $\underline{v} \neq 0$, then

$$h > \frac{2r}{\underline{v}}. \tag{5.18}$$

Let r_k and γ_k be the values for r and γ, respectively, at the k-th discrete transition for $k \in \{0, 1, 2, \cdots\}$. Then for any $\delta \in \mathbb{R}^+$ which defines a $\mathcal{B}_\delta(x_0)$ around a given initial state x_0, $r_0 = \delta$ and $\gamma_0 = 0$. If we consider a γ increment of a radius r at each discrete transition, then, for $k \geq 1$, we have

$$r_k = \delta + \sum_{i=1}^{k} \gamma_i. \tag{5.19}$$

Notice that $\gamma_k + r_{k-1} \geq 3r_{k-1}$ as shown in Figure 5.2(b), to satisfy the condition of $\mathcal{D}_\tau(\mathcal{B}_{r_{k-1}}(x_t)) \subset \mathcal{B}_{\gamma_k + r_{k-1}}(x_t)$ for all $\tau \in [0, h]$. If we choose $\gamma_k := 2r_{k-1}$ for all $k \geq 1$, then

$$r_k = r_{k-1} + \gamma_k = r_{k-1} + 2r_{k-1} = 3r_{k-1} = 3^k \delta. \tag{5.20}$$

From this, (5.18) can be rewritten as follows.

$$h > \frac{2r_{k-1}}{\underline{v}} = \frac{2 \cdot 3^{k-1}\delta}{\underline{v}}. \tag{5.21}$$

However, for a given $\epsilon \in \mathbb{R}^+$, since a sampling period h has an upper bound $h < \epsilon/\bar{v}$ as shown in (5.1), we now have the following inequality:

$$\frac{2 \cdot 3^{k-1}\delta}{\underline{v}} < \frac{\epsilon}{\bar{v}}. \tag{5.22}$$

From this, we finally obtain an upper bound for the number of discrete transition k for given ϵ and δ:

$$k < \frac{1}{2}\log_3\left(\alpha\frac{\epsilon}{\delta}\right) \tag{5.23}$$

where $\alpha := 2\underline{v}/\bar{v}$.

As shown in (5.23), the number of discrete transitions that can be determined by the proposed algorithm is upper bounded by the logarithm of the ratio of ϵ/δ. As an example, for $\epsilon = 1$ and $\delta = 10^{-7}$, k is limited by roughly 7.335 when α is set to 1. This implies that the algorithm can only determine at most the times of seven discrete transitions. Here, it is important to notice that even though (5.23) may not be a precise upper bound for k of the proposed algorithm, it is still useful since it clearly exposes a problem with respect to the algorithm that needs to be resolved. In the next section, we propose one approach to handle this issue.

A Technique for Tight Over-approximation of Discrete Transition State

In this section, the goal is to find a smaller value of γ to construct a tighter over-approximation of a discrete transition state. For this, we suppose that a discrete transition from a location l_i to some other location l_j has already been determined by `IsTransition`$(\cdot)$ function for given h, $\mathcal{B}_r(x_t)$, and $\mathcal{D}_h(\mathcal{B}_r(x_t))$ at some time t, as shown in Figure 5.3(a). Then the procedure for construction of a tight over-approximation of a discrete transition state $x(\tau'; x_t)$ for some $\tau' \in (0, h)$ is as follows:

1. Partition the interval $[0, h]$ into a finite sequence of $\{I_m\}_{m=1}^{M}$ for some $M \in \mathbb{N}$, where $I_m := [(m-1) \cdot \Delta h, m \cdot \Delta h]$ for some $\Delta h \ll h$.

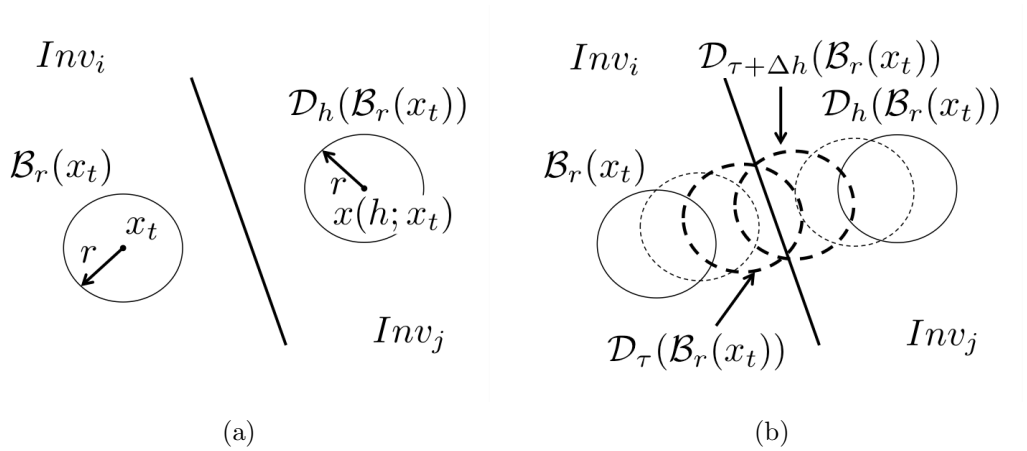


Figure 5.3: Tight over-approximation of a discrete transition state.

2. Find a time $\tau := m \cdot \Delta h \in (0, h)$ such that

   (a) $volume(Inv_i \cap \mathcal{D}_\tau(\mathcal{B}_r(x_t))) > volume(Inv_j \cap \mathcal{D}_\tau(\mathcal{B}_r(x_t)))$, and

   (b) $volume(Inv_i \cap \mathcal{D}_{\tau+\Delta h}(\mathcal{B}_r(x_t))) < volume(Inv_j \cap \mathcal{D}_{\tau+\Delta h}(\mathcal{B}_r(x_t)))$.

3. Construct $\mathcal{D}_{\tau+\Delta h}(\mathcal{B}_r(x_t), \gamma')$ where $\gamma' > \Delta h \cdot \bar{v}$.

4. Compute $\hat{\mathcal{J}}_{i,j} := \mathcal{D}_{\tau+\Delta h}(\mathcal{B}_r(x_t), \gamma') \cap Inv_i \cap Inv_j$.

5. The rest of the procedure is the same as described in `IsTransversal`$(\cdot)$.

To investigate the upper bound on the number of discrete transition for given ϵ and δ, we apply $\gamma_i := 2\Delta h \cdot \bar{v}$ in (5.19). Then we have

$$r_k = \delta + \sum_{i=1}^{k} \gamma_i = \delta + (2\Delta h \cdot \bar{v})\, k. \tag{5.24}$$

If we use the upper bound and lower bound for h as in (5.22),

$$\frac{2r_{k-1}}{\underline{v}} = \frac{2\left(\delta + (2\Delta h \cdot \bar{v})\,(k-1)\right)}{\underline{v}} < \frac{\epsilon}{\bar{v}}. \tag{5.25}$$

From this, we now have the following inequality for the upper bound on the number of discrete transitions with tight over-approximation of a discrete transition state:

$$k < \frac{\delta}{2\Delta h \cdot \bar{v}} \left\{ \frac{\alpha}{4} \frac{\epsilon}{\delta} - 1 \right\} + 1, \tag{5.26}$$

where α is the same as in (5.23).

Unlike (5.23), the upper bound for k in (5.26) is less dependent on the ratio of ϵ/δ as this ratio increases. However, the number of discrete transitions that can be determined through the tight over-approximation of a discrete transition state highly depends on the size of the partition Δh. It is reasonable that the number of discrete transitions that can be detected increases as the value for Δh decreases.

5.2 Evaluation of Bounded ϵ-Reach Set Computation

We first describe an example DTLHA that is used to demonstrate the capabilities of the implementation for a bounded ϵ-reach set computation.

5.2.1 An Example DTLHA: Switching LTI Systems

Consider a closed and bounded continuous state space $\mathcal{X} := [-8, 8] \times [-8, 8] \subset \mathbb{R}^2$. Then we consider an LHA $\mathcal{A}$ that is defined as $\mathcal{A} := (\mathbb{L}, Inv, A, u)$ where

(i) $\mathbb{L} = \{Up, Down, Left, Right\}$,

(ii) $A(l)$ and $u(l)$ for each location $l \in \mathbb{L}$ are defined as shown in Table 5.1, and the vector field for each of these LTI systems is shown in Figure 5.5, and

(iii) $Inv(l)$ for each location $l \in \mathbb{L}$ is defined as shown in Figure 5.4. The mathematical definitions of each of the invariant sets are given in Table 5.2.

It may be noted that all the LTI dynamics defined in the given LHA $\mathcal{A}$ are asymptotically stable. Moreover, from the invariant sets shown in Figure 5.4, and the vector fields shown in Figure 5.5, it is easy to see that every discrete transition which occurs along the boundary of the invariant set between different locations is deterministic and transversal. Hence the given LHA $\mathcal{A}$ is in fact a DTLHA.

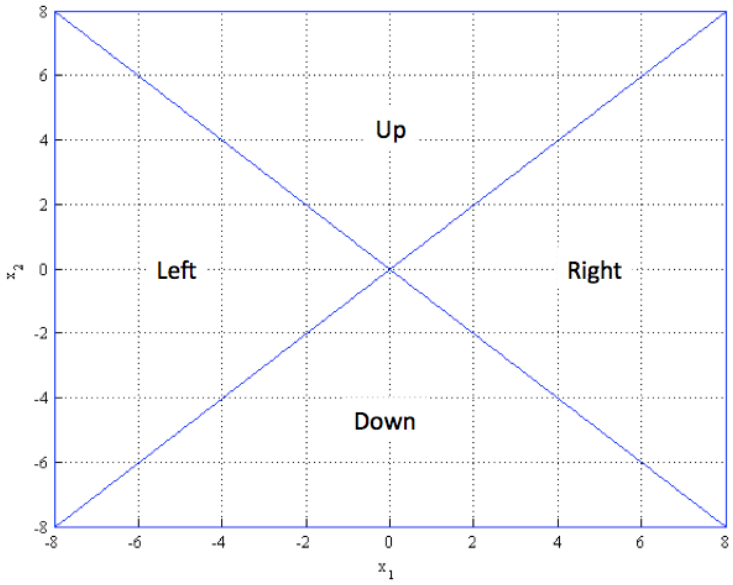


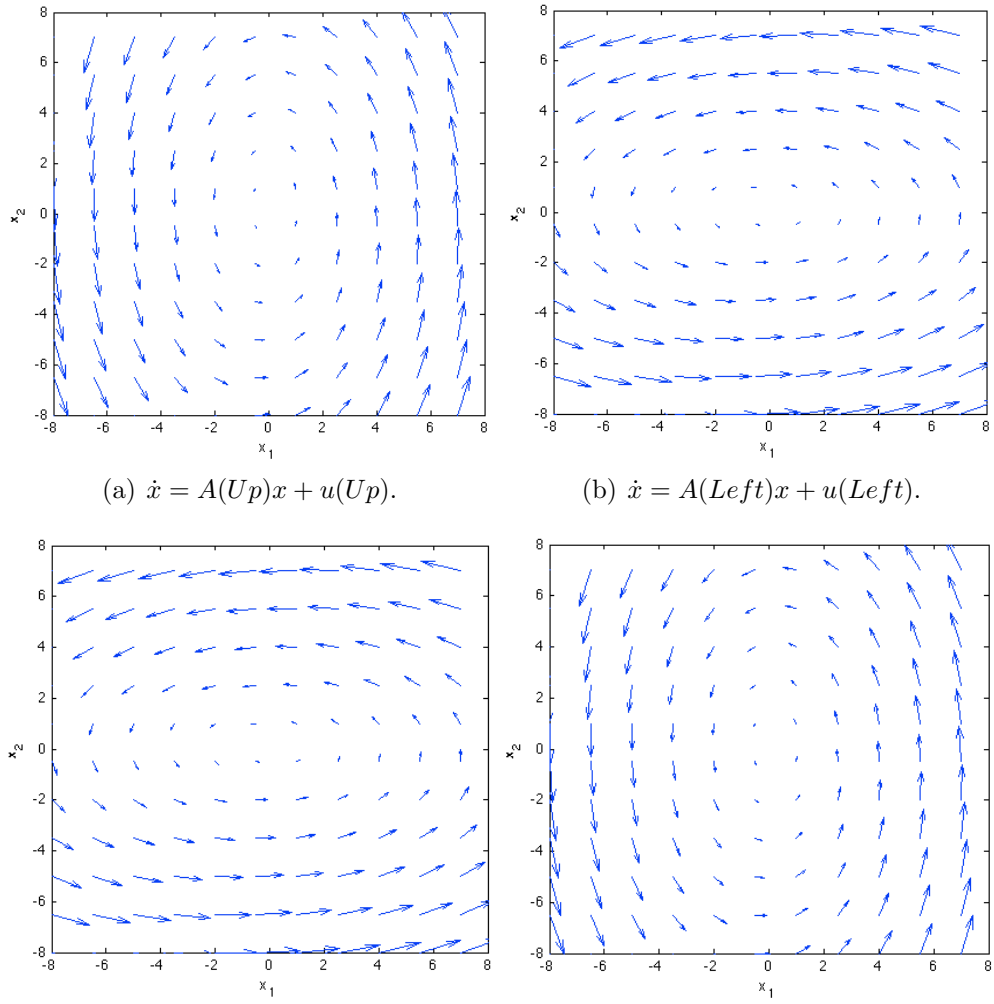
Figure 5.4: Invariant set of each location of the given example DTLHA $\mathcal{A}$.

Table 5.1: $A(l)$ and $u(l)$ for each $l \in \mathbb{L}$ of the example DTLHA $\mathcal{A}$.

	Up	$Down$	$Left$	$Right$
$A(l)$	$\begin{pmatrix} -0.2 & -1 \\ 3 & -0.2 \end{pmatrix}$	$\begin{pmatrix} -0.2 & -1 \\ 3 & -0.2 \end{pmatrix}$	$\begin{pmatrix} -0.2 & -3 \\ 1 & -0.2 \end{pmatrix}$	$\begin{pmatrix} -0.2 & -3 \\ 1 & -0.2 \end{pmatrix}$
$u(l)$	$\begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}$	$\begin{pmatrix} -0.2 \\ -0.2 \end{pmatrix}$	$\begin{pmatrix} 0.15 \\ 0.15 \end{pmatrix}$	$\begin{pmatrix} 0.3 \\ 0.3 \end{pmatrix}$

Table 5.2: $Inv(l)$ for each $l \in \mathbb{L}$ of the example DTLHA $\mathcal{A}$.

l	$Inv(l)$
Up	$\mathcal{X} \cap (x_1 - x_2 \leq 0) \cap (x_1 + x_2 \geq 0)$
$Down$	$\mathcal{X} \cap (x_1 - x_2 \geq 0) \cap (x_1 + x_2 \leq 0)$
$Left$	$\mathcal{X} \cap (x_1 - x_2 \leq 0) \cap (x_1 + x_2 \leq 0)$
$Right$	$\mathcal{X} \cap (x_1 - x_2 \geq 0) \cap (x_1 + x_2 \geq 0)$



(a) $\dot{x} = A(Up)x + u(Up)$.

(b) $\dot{x} = A(Left)x + u(Left)$.

(c) $\dot{x} = A(Right)x + u(Right)$.

(d) $\dot{x} = A(Down)x + u(Down)$.

Figure 5.5: The vector field of the LTI system defined in each location of the given example DTLHA $\mathcal{A}$ where $x := (x_1, x_2)^T$.

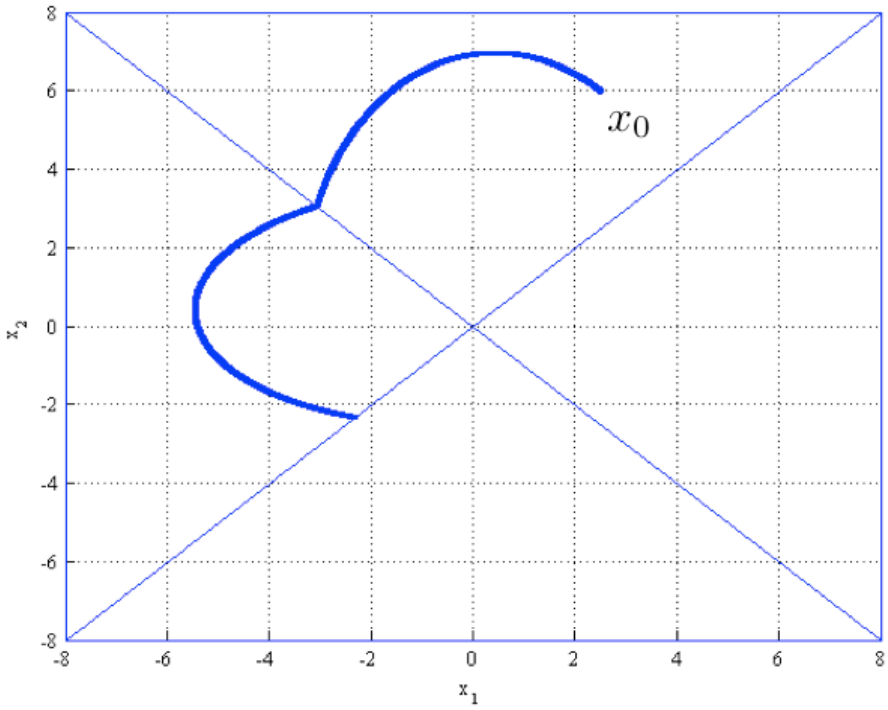
5.2.2 Comparison between Different Values of ϵ

The $\texttt{SystemData} := (\mathcal{X}, \mathcal{A}, l_0, x_0, T, N, \epsilon)$ of this example is the following:

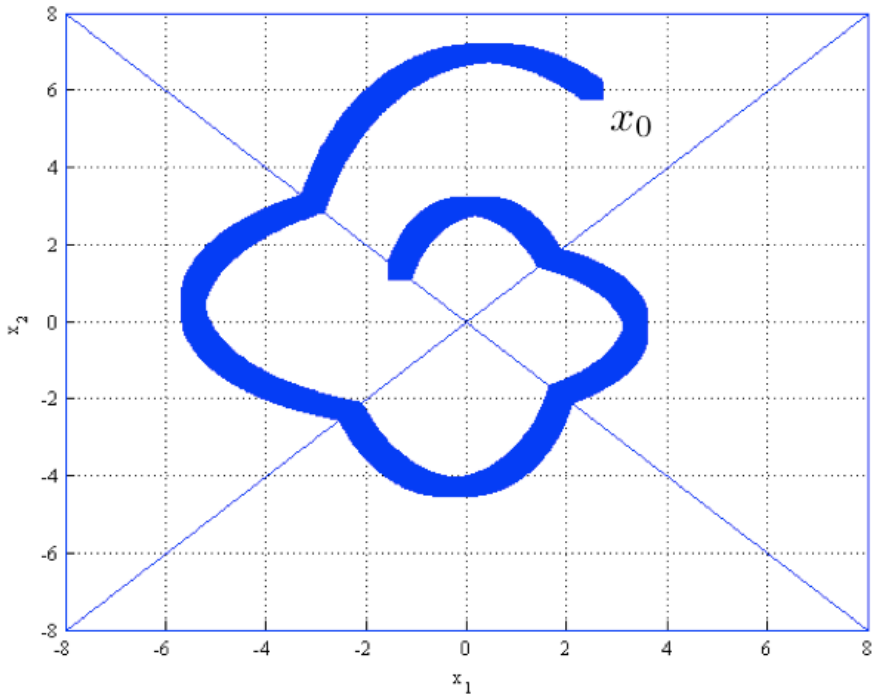
- $\mathcal{X} := [-8, 8] \times [-8, 8] \subset \mathbb{R}^2$,
- $\mathcal{A}$ is as defined in Section 5.2.1,
- $l_0 = Up$,
- $x_0 = (2.5, 6)^T$,
- $T = 10$ sec.,
- $N = 5$, and
- $\epsilon = 0.1$ and 0.5.

To compute a bounded ϵ-reach set of $\mathcal{A}$, we use a policy that (i) uses a fixed value of $\delta = 10^{-5}$ which defines a sufficiently small $\mathcal{B}_\delta(x_0)$, (ii) chooses the value of h and γ on-the-fly to resolve erroneous situations, (iii) chooses k in nondecreasing manner, and (iv) sets a fixed value of 10^{-7} for $\sigma_e, \sigma_i, \sigma_p, \sigma_a, \mu_c$, and μ_h. We also set 10^{-7} as the minimum value for h and γ.

The bounded ϵ-reach sets for two different values of ϵ computed by the implementation are shown in Figure 5.6. For the case of $\epsilon = 0.1$, the algorithm terminates at the computational step $k = 2613$ at which the time $t = 2.2153$ sec. and $\texttt{jump} = 1$ at the location $Left$. The reason for this early termination is that the maximum sampling period h, which is determined by the value of ϵ and the accumulated numerical calculation errors ρ, is not large enough to separate $\mathcal{D}_t(\mathcal{B}_\delta(x_0))$ and $\mathcal{D}_{t+h}(\mathcal{B}_\delta(x_0))$ at the time of discrete transition. Hence the algorithm fails to determine the discrete transition from the location $Left$ to the location $Down$. On the contrary, the algorithm successfully returns a bounded ϵ-reach set of $\mathcal{A}$ for the case with $\epsilon = 0.5$ as shown in Figure 5.6(b). In this case, the algorithm terminates at the computational step $k = 1364$ at which the time $t = 5.8496$ sec. and $\texttt{jump} = 5$ at the location $Left$.



(a) $\epsilon = 0.1$.



(b) $\epsilon = 0.5$.

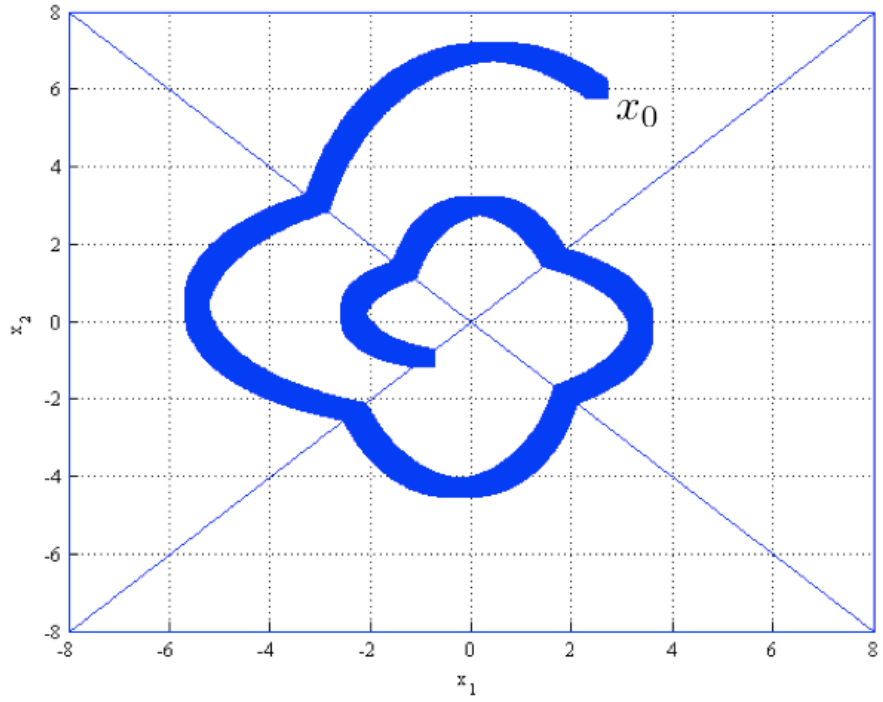
Figure 5.6: A bounded ϵ-reach set of $\mathcal{A}$ with (a) $\epsilon = 0.1$ and (b) $\epsilon = 0.5$.

5.2.3 Comparison between Different Computational Accuracies

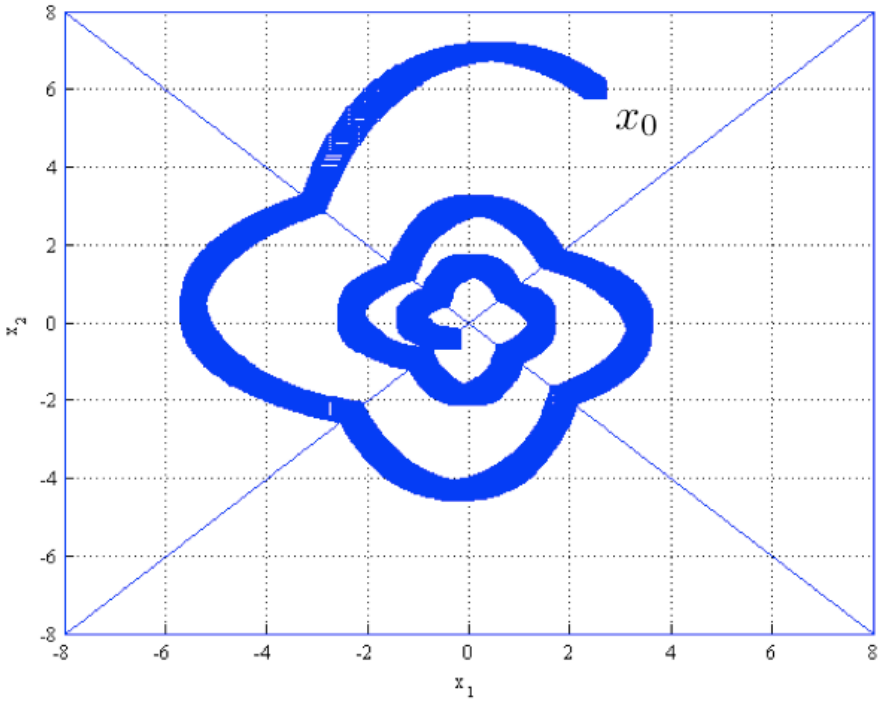
In this section, we compare the results of bounded ϵ-reach set computation of $\mathcal{A}$ between different numerical calculation accuracies to see how does the accuracy affect the computation of a bounded ϵ-reach set. More precisely, we compute bounded ϵ-reach sets of $\mathcal{A}$ when the accuracies (i) 10^{-7} and (ii) 10^{-15} are assumed for the parameters $\sigma_e, \sigma_i, \sigma_p, \sigma_a, \mu_c$, and μ_h, under the assumption that the underlying numerical calculation algorithms used in the implementation support these accuracies. In this example, we use the same `SystemData` as in Section 5.2.2 except (i) $T = 20$ sec., (ii) $N = 10$, and (iii) $\epsilon = 0.5$.

Figure 5.7 shows the computation results for both cases. As evident in the figure, the computation of a bounded ϵ-reach set of $\mathcal{A}$ with 10^{-7} numerical calculation accuracy is not successful in the sense that the computation is terminated before it reaches the normal termination condition that either (i) `jump` $\geq$ `10`, or (ii) $t \geq 20$. In fact, the computation is terminated at the computation step $k = 1659$ when the algorithm tries to determine a discrete transition event from locations $Left$ to $Down$ at the time $t = 7.1209$ sec. and `jump` $= 5$. The algorithm fails to determine this discrete transition event due to the fact that the numerical calculation accuracy used in computing a bounded ϵ-reach set is not sufficiently small enough. Notice that the larger the value of the numerical calculation accuracy the larger the uncertainty on where $\mathcal{D}_t(\mathcal{B}_\delta(x_0))$ is located at time t. Moreover, the numerical calculation error is accumulated as the computation step increases. As explained in Section 4.2, the algorithm computes $\mathcal{D}_t(\mathcal{B}_\delta(x_0), \rho)$ to eliminate this uncertainty where ρ is the accumulated numerical calculation accuracy at time t. Hence, if the value of ρ is too large at the time of the discrete transition, the algorithm cannot make a decision about the discrete transition event since $\mathcal{D}_t(\mathcal{B}_\delta(x_0), \rho)$ and $\mathcal{D}_{t+h}(\mathcal{B}_\delta(x_0), \rho)$ are not well separated across the boundary of the invariant sets. From the data recorded in `ReachSetHistory` of this computation, the value of the accumulated numerical calculation accuracy ρ for $a(\mathcal{D}_t(\mathcal{B}_\delta(x_0)), \rho)$ at the time of termination is 0.0019.

In contrast, as shown in Figure 5.7(b), a bounded ϵ-reach set is indeed returned successfully when 10^{-15} is used as the accuracy with which σ_e, σ_i, σ_p, σ_a, μ_c, and μ_h are determined. In this case, the algorithm terminates



(a) Accuracy = 10^{-7}.



(b) Accuracy = 10^{-15}.

Figure 5.7: A bounded ϵ-reach set of $\mathcal{A}$ with (a) accuracy = 10^{-7} and (b) accuracy = 10^{-15}.

at the computation step $k = 2259$ right after the algorithm makes the tenth discrete transition from locations $Left$ to $Down$ at the time $t = 12.1415$ sec. and `jump` $= 10$. The accumulated numerical calculation error ρ at this termination time is 2.5638×10^{-11}.

5.3 Concluding Remarks

In Chapters 2, 3, 4, and 5 we have addressed issues related to the safety verification problem of cyber-physical systems. We have identified a class of hybrid automata, called Deterministic Transversal Linear Hybrid Automata (DTLHA), for which we can compute a bounded ϵ-reach set. We also have developed a theoretical framework which shows the possibility of the computation of a bounded ϵ-reach set of a DTLHA.

Furthermore, we have addressed the issue of computing a bounded ϵ-reach set of DTLHA with subroutines that only provide finite precision elementary computations. We also have proposed an architecture which separates the elementary subroutines from the policies that adapt the various parameters. This makes the overall algorithm flexible and amenable to different optimizations. An example DTLHA is considered to evaluate the capability of a prototype implementation of the proposed bounded ϵ-reach set algorithm.

Now we move on to the mechanism end of cyber-physical systems; we will specifically be concerned with the design and implementation of a middleware platform for networked control systems.

CHAPTER 6

A MIDDLEWARE FOR NETWORKED CONTROL SYSTEMS

A networked control system is a control system that operates over a distributed system which involves the features of both *heterogeneity* and *distributed operation*. Since these two fundamental features of distributed systems add a lot more complexity to the application, it is quite challenging to develop a networked control application in general. Therefore, it is important to have a simpler abstract model of the system which hides all the complex details of the underlying system from an application developer. A middleware framework can provide such an abstraction of the system to the application developer so that she can develop an application easily on top of the abstraction. In this way, a networked control application can be developed more rapidly and reliably. In the process of designing and developing such a middleware, it is important to consider the domain requirements which capture all the characteristics of the application domain. Thus, as a first step toward the development of the middleware for networked control systems, it is necessary to understand the fundamental characteristics of networked control systems and then establish corresponding requirements for the middleware framework.

6.1 Networked Control Systems

In this section, we first investigate the characteristics of a networked control system. Then based on these domain characteristics, we identity the requirements for a middleware framework for a networked control system.

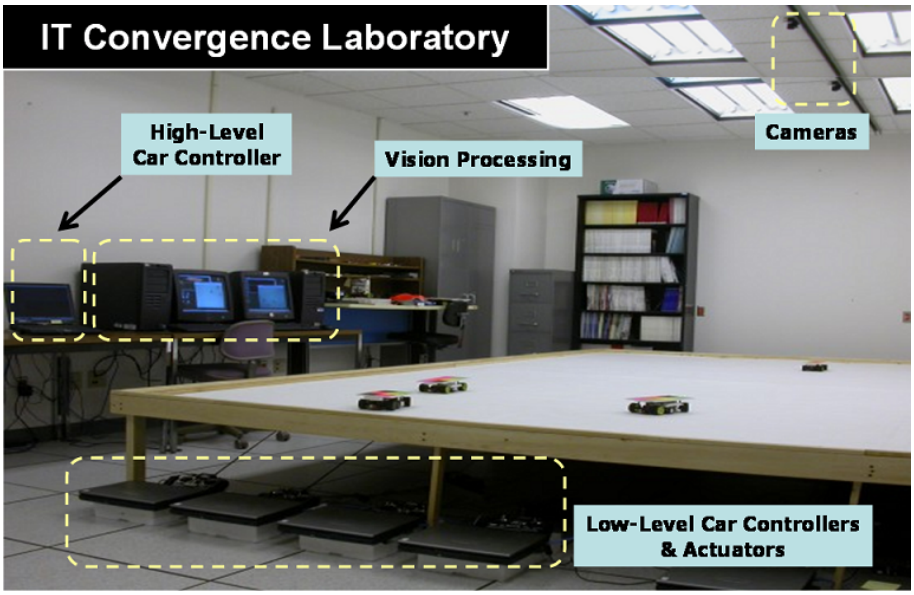


Figure 6.1: Testbed in IT Convergence Laboratory.

6.1.1 Application Domain Characteristics

There are many potential examples of networked control systems in various application areas, such as smart power grids, intelligent traffic control systems, and automatic warehouse management systems. In this section, we investigate the common characteristics which are shared by many networked control systems in many application domains.

Large-scaleness In a networked control system, the control loop is typically formed through the underlying communication network. Thus, the physical distance between the entities in the loop is not the limiting factor anymore. Also, the communication network allows us to form multiple control loops through it so that multiple physical entities can be controlled and control objectives can possibly be achieved at the same time. The Testbed shown in Figure 6.1 is a good example which has such characteristics. In it, a vision system is used to detect the state of a moving vehicle, and a low-level controller controls the vehicle to ensure that it follows a trajectory generated by a high-level controller. The inner control loop for tracking the given reference trajectory is formed through a communication network, since the elements comprising it are running at different computing nodes. Besides this inner control loop, there is another control loop formed through

the same communication network to achieve a slightly different higher level control objective, which is collision avoidance between vehicles. In addition to all these, we also have yet another control loop in the testbed for runtime system management, such as upgrading or migrating some software modules to optimize the overall system performance.

Openness Unlike a classical control system, as noted above in the reference to migration, the distributed control system can change its configuration at runtime. An entity can join or leave the system at runtime. An existing entity can be replaced or migrated to another location at runtime. Also, in some cases, the information flow among the entities constituting a control-loop can be altered dynamically depending on the system states. The testbed in Figure 6.1 possesses all of these open system features. Cars can dynamically join or leave the traffic control system. A controller of a car can be upgraded or even migrated to a better location for better control performance at runtime. If multiple sensing systems are available for controllers to obtain the position/orientation of each car, the controller can dynamically change the selection of the sensor that it subscribes to, depending on the quality of the data.

Time-criticality In most cases, a control system is a time-critical system in which a given action is required to occur at the right time. Failure to do so can degrade the system's performance or even can cause the system to become unstable. Delay induced oscillations are one low-level example of this. As another example, in the testbed, two cars might collide with each other if they are not controlled in a timely and coordinated fashion.

Safety-criticality A safety-critical system is one in which the cost of system failure is very expensive, causing severe damage or harm to people, equipment, or the environment. Many control systems are indeed safety-critical systems since they typically operate on physical systems.

6.1.2 Operational Domain Requirements

Following from the application domain characteristics, we now identify some of the operational domain requirements. The following are some of the re-

quirements for a middleware framework for networked control systems.

Operational Requirements In a distributed application, entities comprising a control application typically run on different computing nodes, with each of the computing nodes having a different clock. The fact that the overall system is distributed over a communication network makes it potentially much harder or more time-consuming to develop an application. Also, clock differences between computing nodes accentuate difficulties in developing a distributed control application. Therefore, it is necessary to have some mechanisms in a middleware framework that can resolve the problems of both *location difference* and *time discrepancy*. Besides these two requirements, a mechanism which supports *semantic addressing* (or context-aware addressing) is also a desirable feature since it can significantly improve the portability and reusability of the application code.

Management Requirements Owing to the open system feature, a distributed application is typically subject to change after its deployment. Sometimes an entity needs to be either changed or migrated, as noted earlier. Entities can dynamically join or leave the application configuration at any point of time. However it is not always possible to stop the whole system for these changes. Therefore, it is necessary that a middleware framework provides mechanisms for *runtime system management* which allow continuous system evolution.

Non-functional Requirements The non-functional requirements [1] for a middleware framework are induced from both the time-critical and safety-critical characteristics of a networked control system. The time-criticality requires a control system to behave in a predictably timely manner so as to minimize the effect of delay. Thus, a middleware framework is required to provide some mechanisms which support the *timeliness* behavior of the control system. Also, the safety-criticality of a control system requires that

[1] It should be noted that the phrase "non-functional requirements" can be used in dramatically opposite ways in different communities: with respect to the middleware designer, both a naming service or communication mechanism are both functional requirements, but achieving control loop stability is a non-functional requirement. From the viewpoint of the control designer, the reverse is true. In this paper, the viewpoint is that of the middleware designer.

the middleware framework itself be error-free, and also provide some mechanisms to tolerate faults that can occur in the application layer, so as to achieve overall *reliability*.

6.2 Etherware

Etherware is a middleware framework for networked control systems that has been developed at the University of Illinois [1]. In this section, we discuss how Etherware is designed and how it works to support the domain requirements established above from the domain characteristics.

6.2.1 Domainware for Networked Control Systems

The main objective of Etherware is to provide a software framework which enables a rapid, reliable, and evolvable networked control application development. A networked control application can be easily developed in Etherware since it supports *component-based application development*. A software component can be thought of simply as a software module which provides a set of functions through a set of interfaces. One major benefit of component-based programming is that an application can be developed easily owing to the composability of components. Components can interact with each other through either method invocation or a message exchange mechanism. Etherware uses the message exchange mechanism for component interaction. In Etherware, *Message* is a well-defined XML document object and it is the root class in the hierarchy of the Message class. A new message type for applications can be easily defined by extending the Message class. Listing 6.1 shows the XML structure of the Message class.

Listing 6.1: XML structure of an Etherware Message

```
<EtherMsg type=... rel=... >
  <profile name=... > </profile>
  <content> ... </content>
  <ts value=... > </ts>
</EtherMsg>
```

A name of the message can be specified in the `type` attribute of the `EtherMsg` element. In the `profile` element, the name of the recipient component is specified. In the `content` element, any information concerning the interaction semantics can be specified. The clock time when the message is created is specified in the `ts` element.

6.2.2 Architecture

Etherware is designed based on the concept of microkernel architecture in operating systems [24]. Roughly, Etherware consists of a *Kernel* and *components* as shown in Figure 6.2. Components can be classified further into *service components* and *application components*. The Kernel provides a set of fundamental functionalities for middleware operations, such as component life-cycle management and message delivery among components. To deliver a message from one component to another, Kernel creates and uses a *job*. A job is a scheduling entity in Etherware which contains the message to be delivered and the address of the recipient component. When a new message arrives in the Kernel, the Kernel encapsulates the message into a job and enqueues it into a job queue. The jobs in a job queue are processed one by one by a job processing software module, called *Dispatcher*. The other functionalities which are required to be implemented in a middleware framework are provided as several service components. Section 6.2.4 discusses services in more detail.

6.2.3 Component Model

In Etherware, as noted above, an application can be developed as a set of components. Etherware's component model, shown in Figure 6.3, provides the framework in which an application component can easily be developed. In designing the component model, several *software design patterns* [25] are used. *Shell*, a class object whose design is based on the *Facade* software design pattern, encapsulates a user-defined class object which implements the application logic. It manages the life-cycle of the encapsulated class object and provides an interface which allows the encapsulated class object to interact with the other components. The *Strategy* design pattern is used to

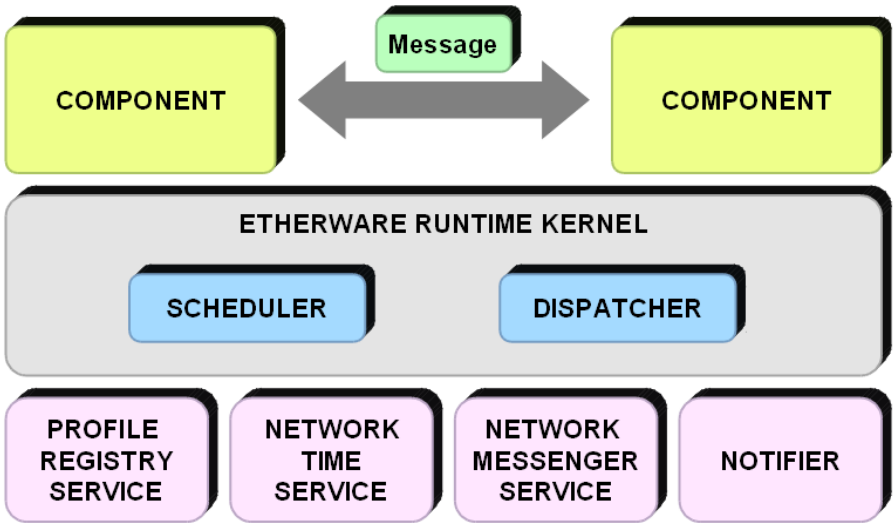


Figure 6.2: Etherware architecture.

design a uniform interface between Shell and the class object encapsulated by it. Due to this Strategy design pattern, Shell can be used to perform runtime *component replacement*. For *component migration*, the execution state of a component should be smoothly continued after the migration to avoid disrupting performance. The *Memento* design pattern was adopted to support this feature.

6.2.4 Services

Etherware supports several fundamental functionalities which are commonly required for networked control system applications as Etherware services. *ProfileRegistry* is a naming service that is implemented in Etherware to support the semantic addressing requirement. It maintains information about the profile of a component and its network address. *NetworkMessenger* provides the service of message delivery over the network. It encapsulates all the details about network information such as protocol and network address. NetworkMessenger is therefore the Etherware service which supports the domain requirement of hiding location discrepancy. NetworkMessenger is called only when a message is destined for a remote component, since Etherware Kernel delivers the message directly if it is for local component. Etherware resolves the time discrepancy issue of distributed systems by implementing

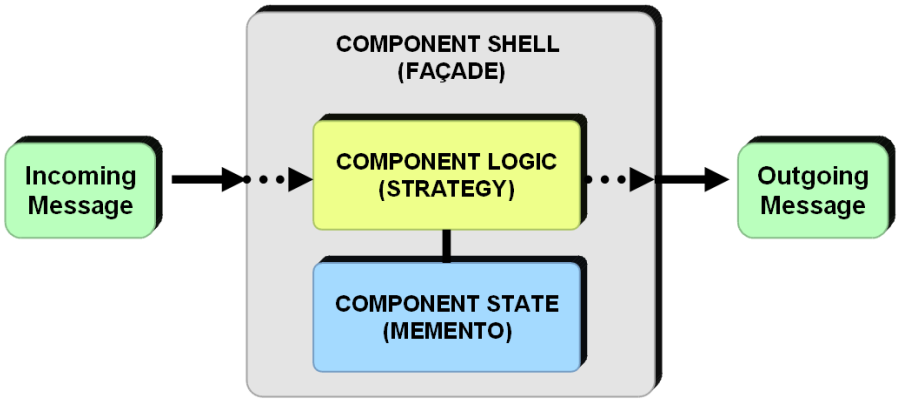


Figure 6.3: Etherware component model.

the *NetworkTime* service. NetworkTime service translates a time stamp from the clock of a remote computing node to that of a local machine for every message that is received by NetworkMessenger. For this purpose, NetworkTime service maintains the clock offset and skew for every other computing node where another NetworkTime service is running, by periodically exchanging *Ping* and *Response* messages. The *Notifier* service provides a time-triggered message service to Etherware. Basically, Etherware is an *event-driven system* such that a component gets executed only when it receives a message. However, in many cases, control actions need to be performed based on the *time*. In such situations, the Notifier service enables a component to execute at the time that it has to, by sending a notification message to that component.

6.3 Analysis of Domain Requirement

Table 6.1 compares the domain requirements and the functionalities supported by Etherware [1]. As noted in previous sections, Etherware already satisfies many important domain requirements which provide flexibility. However, it still needs additional features to satisfy requirements which are essential for control systems, shown in Table 6.1, key among which is *timeliness*. This requires the addition of appropriate real-time mechanisms that can be used to provide timeliness guarantees, which is the subject of the next section.

Table 6.1: Domain requirements vs. Etherware implementation.

Domain Requirements		Etherware Implementation
Operational	Location transparency	NetworkMessenger
	Hiding time discrepancy	NetworkTime
	Semantic addressing	ProfileRegistry
Management	System evolution	Component model
Non-functional	Timeliness	
	Reliability	

CHAPTER 7

ETHERWARE MECHANISMS FOR REAL-TIME GUARANTEES

In this chapter, we address the design of several critical enhancements to support the *timeliness* requirements of control systems. The timeliness requirement is especially important, since in many control systems, the performance of the system could be degraded or the stability of the system could itself be compromised if the sensing and control actions are not executed at the right time.

7.1 Issues for Real-Time Properties

A real-time system is not a system which is fast, but is rather a system which is predictable. In fact, *predictability* is one of the most fundamental attributes of any real-time system [26]. What one basically requires is that the system[1] should behave in such a way that the execution behavior of the running task set can be precisely described from the information about both the system itself and the task set. In general, predictability depends on every aspect of the system, including hardware platform, communication network, operating system, programming language, etc. A middleware is a software framework interposed between the operating system and application programs. Therefore, in the following discussion, we assume that the hardware and software over which a middleware is executed comprises a predictable real-time platform. Examples of such platforms are computer systems running a real-time operating systems such as VxWorks [27], QNX [28], and Real-Time Linux [29].

[1] In this context, by "system" we mean the computing system where computational tasks are executed.

7.2 Design of Etherware Scheduling Mechanism for Real-Time Guarantee

Real-time scheduling is the very first requirement for supporting predictability. Many different types of Quality of Service (QoS), ranging from static (e.g., period) to dynamic attributes (e.g., deadline), are used for real-time scheduling. The scheduling policy itself can be arbitrarily complex and have multiple layers of scheduling decisions using many types of QoS. In this work, we adopt the concept of *hierarchical scheduling* [30] to support an arbitrary scheduling policy. The main idea behind this design is that at the first stage the scheduler schedules jobs based on some *static QoS*, so that it can classify the static class, and then, at the second stage, the scheduler schedules the jobs within the same static class using some *dynamic QoS*. By combining these two static and dynamic scheduling hierarchies, it is possible to realize many scheduling policies from complete static at one extreme to complete dynamic at the other.

7.3 Quality of Service of Message Delivery

In this work, we define Quality of Service (QoS) as a collection of attributes of an application that are used in scheduling for execution. The QoS specification can contain arbitrary types of attributes which affect the execution of an application. As shown in Listing 7.1, it could be the period, the relative deadline, or the worst case execution time (wcet) of an application execution, or it could be an attribute related to the application's importance in the task set.

Listing 7.1: QoS specification in Message.

```
<EtherMsg type=... rel=... >
  <profile name=... > </profile>
  <content> ... </content>
  <ts value=... > </ts>
  <QoS crit=... period=... deadline=... wcet=...>
  </QoS>
</EtherMsg>
```

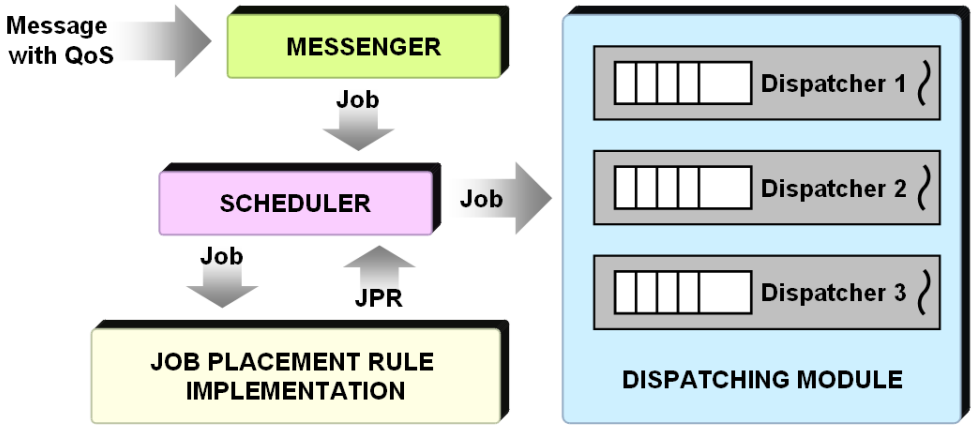


Figure 7.1: Real-time scheduling mechanism with three Dispatchers.

Once we define a set of attributes as a QoS requirement, then the question is: Where does this QoS specification need to be embedded so that it can be used in Etherware's scheduling action? Considering the fact that Etherware is an event-driven system, as described in Section 6.2.4, the Message class object is the right place to put QoS. Therefore, the XML document of a Message class object is modified to contain an element called `QoS`, which has attributes of `crit`, `period`, `deadline`, and `wcet` as illustrated above. Now, an application component can specify its QoS information about message delivery whenever it creates and sends a message to other components. Then Etherware uses the specified QoS information when it makes a scheduling decision for message delivery.

7.4 Priority-based and Concurrent Scheduling

Most real-time operating systems support priority-based scheduling ([27], [28], [29]). They provide predictable behavior with respect to the priority of a process (or a thread in some cases). This means that if a process has higher priority than other processes in the system, then it gets to be executed. Etherware utilizes this priority-based scheduling mechanism of the underlying platform to render it a real-time middleware.

Even though the existing Etherware Kernel does not support concurrent message processing, it has a mechanism such that it can be easily extended

to have a concurrent processing module. As explained in Section 6.2.2, Dispatcher is a software module, inside Etherware Kernel, for processing a job. For concurrent processing, we have therefore added to Etherware Kernel a *dispatching module* consisting of multiple Dispatchers, as shown in Figure 7.1. It is worth mentioning that the number of Dispatchers in the dispatching module is not fixed. The Etherware mechanism for the dispatching module is designed such that the decision about the number of Dispatchers and the priority[2] of each Dispatcher can be specified by an Etherware user. This specification (or *policy* in more formal terminology) is called a *Thread Scheduling Rule* (*TSR*) in Etherware. Each Dispatcher has its own job queue to hold jobs to be processed by the Dispatcher, as shown in Figure 7.1. Now, the job queue is modified to become a prioritized queue so that jobs in the queue can be ordered based on a specified attribute of each job.

Typically, a scheduler makes a decision about execution order among tasks. However, the Etherware Scheduler shown in Figure 7.1 operates a little differently from such typical scheduling actions. Instead of deciding an execution order among tasks, it determines the right place where a job (a scheduling entity in Etherware Kernel) should go in the dispatching module. When the Etherware Scheduler makes a decision, it refers to and implements the rules specified by an Etherware user. This user-specified rule is called the *Job Placement Rule* (*JPR*). To find a right place in the dispatching module, Etherware Scheduler needs two pieces of information, one to select a Dispatcher in dispatching module and the other to find the right position in the job queue of the selected Dispatcher. Thus, a JPR returned from a user-implemented software module as shown in Algorithm 4 should contain such information.

7.5 Issues Concerning Implementation

For better predictability, the concurrent message processing mechanism is adopted, as explained in Section 7.4. One of the major concerns that needs to be addressed in any concurrent system is *synchronization*. Without having

[2]The specific priority set that is supported is dependent on and given by the underlying software platform.

Algorithm 4: A pseudo-code example of Job Placement Rule implementation.

Input: job := (Message, QoS, Recipient's Address)
Output: jpr := (Dispatcher ID, Dynamic Priority)

```
now = currenttime()
if criticality(job) = high then
 | jpr ← (1, now + period(job))
else if criticality(job) = middle then
 | jpr ← (2, now + period(job))
else if criticality(job) = low then
 | jpr ← (3, now + period(job))
end
```

proper synchronization among multiple concurrent processing entities[3] in a software program, the program might fall into a *deadlock* situation or produce some unexpected execution outcomes due to a *race condition* [24]. Therefore, as a first step toward a concurrent real-time software system, it is always an important procedure, from the implementation point of view, to analyze and modify the software system to make sure that it has the right synchronization at every required place. This is indeed the source of difficulty for concurrent programming.

Etherware was originally developed using the *Java* programming language [31], for several reasons. First, it is easy to develop a software program over a well-designed Object-Oriented Programming (OOP) language. Java has many built-in class packages for networking, data structures, multiprocessing, and so on. Also, it releases the burden of memory management from a programmer, through its garbage collection mechanism. Second, it supports platform independent application development. Once a software program is written, there is no need to modify the code to port it onto another operating system platform. Therefore, from the software engineering point of view, Java is a good candidate for developing a software program for a distributed system such as Etherware.

However, there are a couple of caveats concerning Java, with respect to real-time performance. Java was not originally designed for real-time applications. It is designed and optimized for performance in terms of overall

[3]This processing entity is typically called a *process* or a *thread* in operating systems.

throughput rather than predictability. Specifically, the dynamic loading, linking and initialization of classes or interfaces of Java Virtual Machine (JVM) could cause unpredictable delays in program execution. Even worse, unpredictability comes from the runtime memory management, i.e., garbage collection. Thus, even though Java is well suited for a distributed application, it is not well suited for a distributed control application. However, responding to the recent increased demands for real-time embedded systems, there have been several efforts to expand Java's application domain into the real-time computing areas over the past decade. Toward this goal, the first version of the Real-Time Specification for Java (RTSJ) [32] was released in 2000 through the Java Community Process (JCP). In 2005, Sun Microsystems released its first version of the RTSJ implementation, called *Sun Java Real-Time System* (*Sun JavaRTS*) [33].

At the heart of the specifications defined in RTSJ for better predictability are specifications for thread scheduling/dispatching and memory management. For real-time execution, two new classes of threads are defined in RTSJ. `RealtimeThread` (RTT) extends the standard `java.lang.Thread` class and also implements the `Schedulable` interface, which is also newly defined in RTSJ. In terms of predictability, RTT can only provide soft real-time performance. A thread that is an instance of RTT can still be preempted by some internal behaviors of JVM, such as garbage collection, dynamic object loading, and so on. For better predictability, RTSJ recommends using `NoHeapRealtimeThread` (NHRTT), which is extended from RTT. Based on RTSJ, NHRTT can provide hard real-time performance under some strict restrictions on memory usage to avoid preemption by a garbage collector. However, this requires significant change to the programming model, due to the restrictions on memory usage, making it much harder to write a program in Java. Therefore, NHRTT is not yet a practical solution for hard real-time performance, especially for a large-scale software program like Etherware. Since we believe that the easy-to-use programming model of current Java is one of its biggest benefits, we do not employ NHRTT to achieve hard real-time performance.

Besides RTSJ, there have been several research works on real-time garbage collection (RTGC) ([34, 35]) to improve predictability of Java programs. In this line of research, the basic objective is to develop a garbage collection mechanism which performs automatic memory management without sacri-

ficing the predictability of RTT. However, even though there has been some significant improvement in terms of predictability with RTGC mechanisms, it is still necessary to use the NHRTT mechanism of RTSJ for hard real-time guarantees.

To implement Etherware's real-time mechanisms, we use the second version of Sun JavaRTS which includes an RTGC enhanced from the work in [34]. With this RTGC, an RTT in Sun JavaRTS provides quite good performance in terms of predictability in most cases, as we demonstrate in the following chapter.

CHAPTER 8

NETWORKED INVERTED PENDULUM CONTROL SYSTEM

When we combine the temporal predictability provided by our real-time enhancement with the flexibility provided by the Etherware design, we generate rather powerful capabilities. In this chapter, we demonstrate two such capabilities, controller upgrade at runtime, and controller migration at runtime on an unstable system. Controller upgrade refers to the capability to change the control law while a system is running. Controller migration refers to the capability to relocate where a control law is being computed, even while the system is running. Etherware makes such flexible design easy, since its mechanisms are designed to be inherently flexible. However, in order to reliably make use of these flexible mechanisms, one needs to ensure that system stability, a property that depends on time-critical actions, is preserved even under upgrade and migration. Our real-time enhancements make possible such stability preservation, as we demonstrate. Thus, the incorporation of these real-time mechanisms with the other flexibility providing functionalities provided by Etherware leads to a more powerful framework for networked control system design.

8.1 Inverted Pendulum Control System

To illustrate the application of our real-time Etherware for networked control, an inverted pendulum system was chosen, since it is an inherently unstable system that requires strict predictability of the computing system on which the controller is executed. Figure 8.1 shows the rotate type inverted pendulum system [36] that is used in the experiment. As illustrated in Figure 8.2, it has two links: one is on the base that is actuated by a DC motor attached to it and the other is a passive link. Also, it is equipped with a DSP board for both measuring the angles of both joints and for applying the PWM signal to

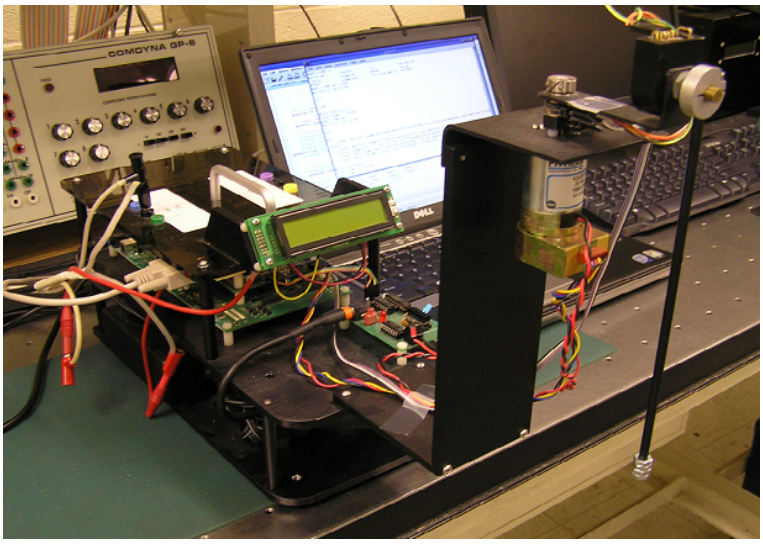


Figure 8.1: Inverted pendulum control system.

a DC motor. The controller actually runs on a PC that is connected with the DSP through RS-232C serial communication. The controller is developed as an Etherware component running on Etherware, which in turn runs on Sun JavaRTS for real-time performance.

In our implementation of the controller, the controller is activated every 15 ms. In each period, the controller first requests the angle data from the DSP. Once it receives the measured angle data, it then computes a control output value and sends it back to the DSP. Then the DSP board delivers the control action right after it receives the control command from a PC over the serial port. Figure 8.3 shows a trace of these periodic interactions between the PC and DSP. The upper signal in the scope image is the signal for feedback of angle data from DSP to PC, and the lower signal is the signal from PC to DSP for requesting angle data and sending a control command. From Figure 8.3, we can see the good periodic behavior of the Etherware controller for its periodic control action, which demonstrates its real-time performance.

8.2 Periodic Control Under Stress

To verify the timeliness guarantee of the real-time Etherware, the inverted pendulum is controlled by a periodic controller while the CPU is subjected to a stress condition. In this experiment, to stress the computing node where

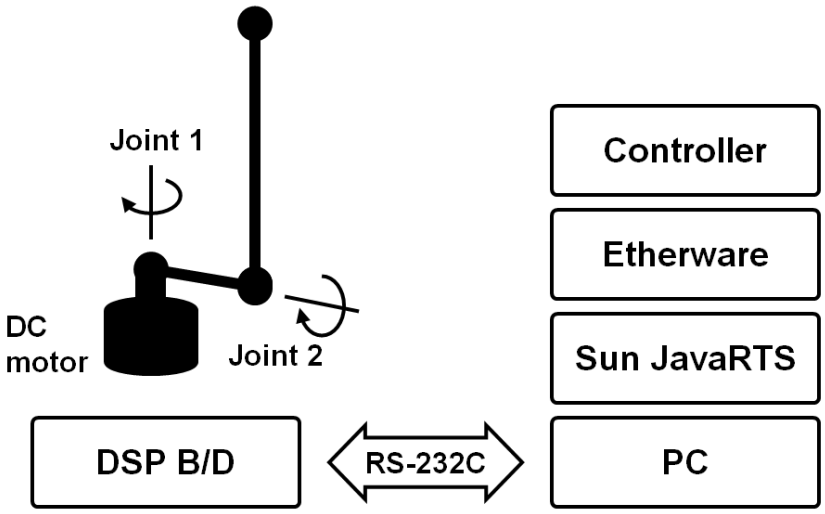


Figure 8.2: Schematic of the inverted pendulum control system.

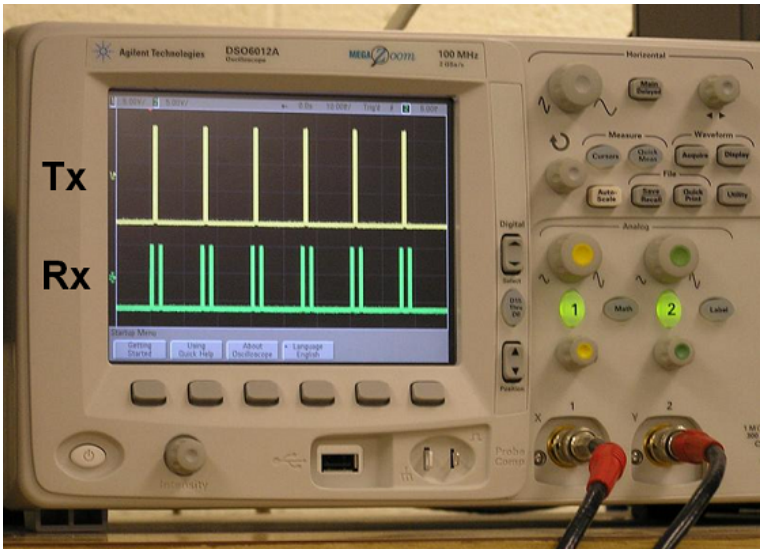


Figure 8.3: Periodic sensing and control over RS-232C communication.

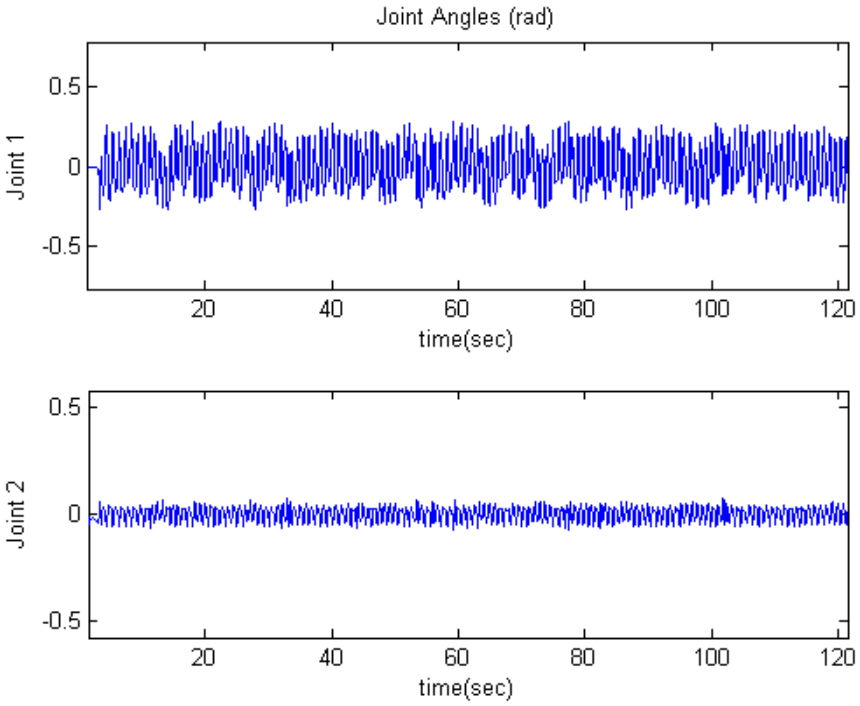


Figure 8.4: Periodic control of an inverted pendulum under stress.

a periodic controller is running, an extra periodic task which consumes approximately 20 percent of the CPU time is made to run concurrently with the periodic control task. The stress task has a period of 5 seconds, and consumes 1 second to execute its computational task. Clearly, the 1 second execution of the stress task is long enough to make the inverted pendulum system unstable if the real-time performance is not supported by Etherware.

To achieve timely execution behavior of the periodic control task, a higher execution priority is assigned to the periodic control task than that assigned to the stress task. The experimental result is shown in Figure 8.4. In this experiment, the stress task starts to execute its periodic task around 20 seconds after the system starts. As shown in the result, the stability of the inverted pendulum has not been affected at all by the execution of the stress task.

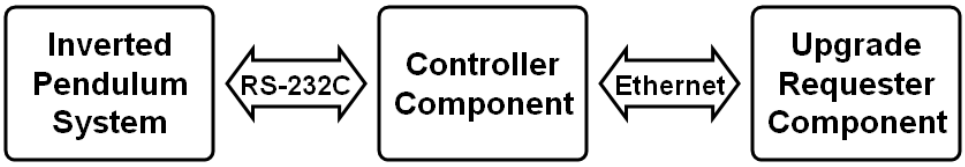


Figure 8.5: System configuration for controller upgrade.

8.3 Runtime System Management

The experiments to be described in this section emphasize the necessity of a real-time middleware framework for networked control systems. The specific capabilities that we aim to provide are controller upgrade and controller migration.

It is important to have predictability concerning the behavior of a networked control system that is subject to such change of its application configuration at runtime, and to determine a priori whether the real-time mechanisms can support the runtime management features in Etherware. In some applications, we can safely perform the system reconfiguration at runtime. However, in some other applications, which are typically time-critical control systems like an inverted pendulum system, it is necessary to have a guaranteed time bound for the time of reconfiguration. Without having such a time bound, the control system may fail to maintain its stability. In the sequel, we detail experimental results concerning the two above-mentioned important runtime system management tasks - controller upgrade and migration.

8.3.1 Controller Upgrade

Here the goal is to upgrade a controller at runtime to change a control law while the system is running. Figure 8.5 shows the configuration of a networked application for such runtime controller upgrade. The controller runs on the PC that is directly connected to the inverted pendulum system through a serial communication. However the component which requests the controller upgrade runs on the other PC and sends its request to Etherware around 30 seconds after the system is started, for better control performance. The angles (0, 0) are the reference positions for the two joints of the inverted pendulum. The joint angles are measured during the controller upgrade pro-

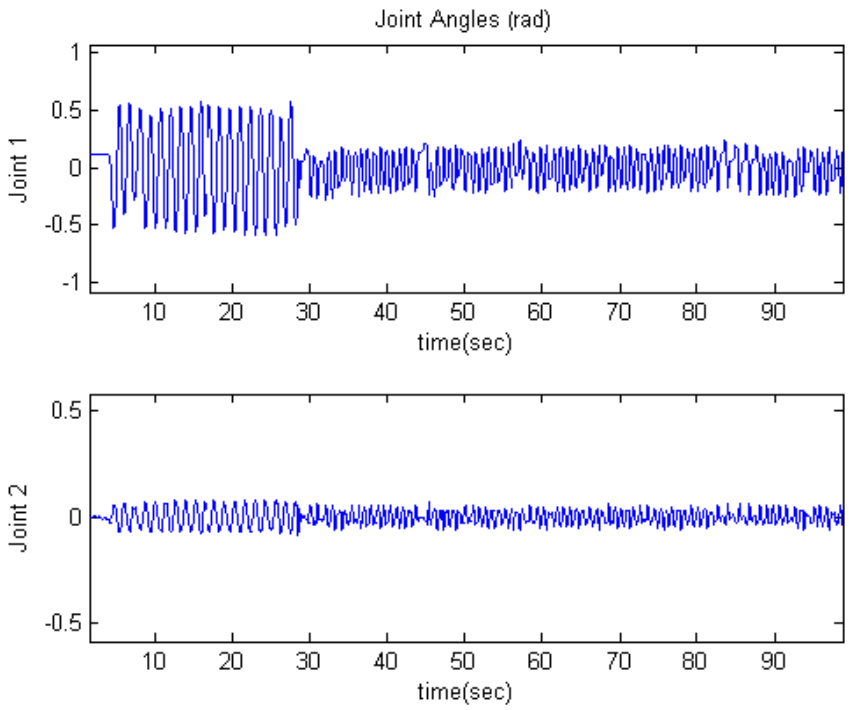


Figure 8.6: Joint angles of the inverted pendulum under runtime controller upgrade.

cess and plotted in Figure 8.6.

As seen in the results, the second link of the inverted pendulum maintains its vertical upright position even while the controller is upgraded. Also, we can see the difference in control performance before and after controller upgrade.

8.3.2 Controller Migration

Here the goal is to move the location where the control law computation is performed from one node to another, while the system is running. Such a capability can be important in optimizing the behavior of control systems. For example, if network delays cause congestion, one may want to relocate where the control law is computed. More generally, one may want to optimize control loop performance with respect to delays. Such outer loops will be important to future design of complex reliable control systems.

To migrate a component from one PC to another PC, Etherware per-

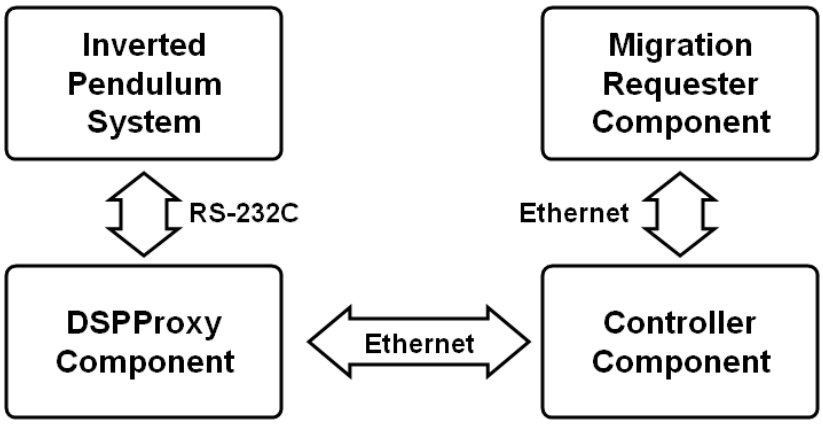


Figure 8.7: System configuration for controller migration.

forms several steps of actions internally, such as the externalization of the component's runtime state, creation of a new instance of component at the destination node, restoring the externalized state in a new component instance, and more. Therefore, component migration is a much more difficult task than upgrade, in general. For *safe* component migration, all these actions are required to be performed in a timely manner in time-critical control systems. As stated in the previous section, this is an important situation in which real-time mechanisms are essential.

Figure 8.7 illustrates the system configuration for a controller migration application. The inverted pendulum is initially controlled by a controller running at a remote computing node in this application. Since the controller cannot directly communicate with the inverted pendulum system, another component, the DSPProxy component, is developed to mediate the interaction between the inverted pendulum system and the periodic remote controller. Yet another component which requests controller migration to Etherware runs at the third computing node. In the experiment, the requester component sends a controller migration request around 40 seconds after the system is started. Then the controller is migrated from the remote node to the node where the inverted pendulum is directly connected.

Figure 8.8 shows the measured joint angles during the controller migration process. As shown in the plot, there is no loss of the stability of the inverted pendulum control system during the migration.

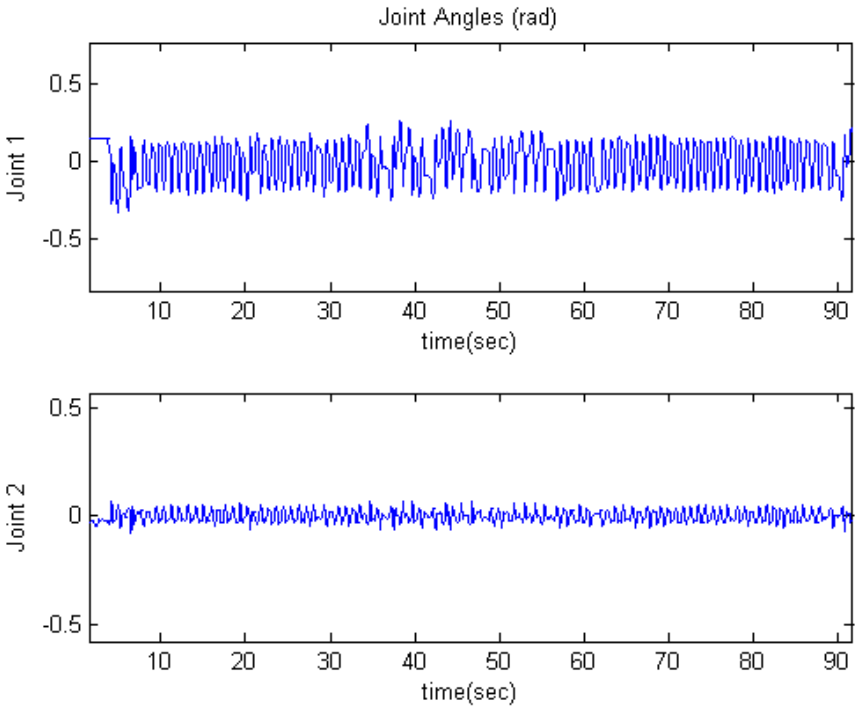


Figure 8.8: Joint angles of the inverted pendulum under runtime controller migration.

8.4 Concluding Remark

In the second part of this thesis in Chapters 6 to 8, we argued that it is indeed important to develop a real-time middleware framework for realization of flexible networked control systems. Since networked control systems are typically open and large-scale systems, we argue that the framework should be designed with an appropriate architecture and mechanisms to support the fundamental requirements of networked control systems. In view of the domain requirements identified in Chapter 6, we have enhanced the initial Etherware of [1] to support the time-critical nature of the domain requirements in Chapter 7. Furthermore, in Chapter 8, we have demonstrated the temporal predictability of our real-time Etherware through the experiments with a networked inverted pendulum control system involving sophisticated runtime functionalities such as runtime controller upgrade and migration.

CHAPTER 9

CONCLUSION

In this dissertation, we have studied cyber-physical systems from two different viewpoints.

In the first instance, we analyzed cyber-physical systems as hybrid systems, since the dynamical behavior of cyber-physical systems typically includes both continuous and discrete dynamics. Through the hybrid system formalism, the safety verification problem of cyber-physical systems reduces to the reachability problem of hybrid systems. However, it is still a challenging problem in general to solve the reachability of hybrid systems even with simple dynamics. Hence, developing algorithms for computing an over-approximation of the reachable set for various classes of hybrid systems becomes one of the important directions in hybrid system verification research.

Pursuing this line of research, we have identified a special class of hybrid systems, called Deterministic Transversal Linear Hybrid Automaton (DTLHA), and shown that, for any $\epsilon \in \mathbb{R}^+$, an ϵ over-approximation of the reachable states of such hybrid systems from a fixed initial state up to a finite time, called a bounded ϵ-reach set, can be computed using infinite precision calculations. We also have shown that a bounded ϵ-reach set of a DTLHA can still be computed without assuming such infinite precision calculation capability even though the approximation size ϵ cannot then be arbitrarily small in this case, since it is inevitably limited by the numerical precision supported by the underlying calculations. Besides this theoretical development, we have also proposed an architecture and an algorithm for a software tool for a bounded ϵ-reach set of a DTLHA and demonstrated the capability of a prototype implementation of the proposed algorithm through an example.

As to the second viewpoint, cyber-physical systems can be viewed as networked control systems, since their constituents consisting of sensors, actuator, and controllers are typically distributed over a communication network

and interact each other through the network. Due to this inherent structural complexity, networked control systems give rise to several new problems which need to be resolved for successful development of such control systems. One of the most important is to develop a middleware with appropriate architecture and mechanisms for rapid, reliable, and evolvable networked control applications.

Towards this end, we have investigated the fundamental characteristics of networked control systems and deduced several key requirements for a middleware for networked control systems domain. Then, we have identified the enhancements still required by Etherware [1], a middleware for networked control system that has been developed in the Information Technology Convergence Laboratory at the University of Illinois. Specifically, the timeliness is a crucial requirement that must be provided by a middleware for networked control systems. To address this problem, we have proposed a real-time message scheduling mechanism of Etherware and evaluated the performance of the enhanced Etherware by applying it to a networked inverted pendulum control application. Through several experiments which involve sophisticated runtime functionalities such as component upgrade and migration, we have validated the temporal predictability and flexible capabilities of the enhanced Etherware.

REFERENCES

[1] G. Baliga, "A middleware framework for networked control systems," Ph.D. dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.

[2] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *ACM Symposium on Theory of Computing*, 1995, pp. 373–382.

[3] J. Lygeros, C. Tomlin, and S. Sastry, "Controllers for reachability specifications for hybrid systems," *Automatica*, vol. 35, no. 3, pp. 349–370, March 1999.

[4] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas, "Discrete abstractions of hybrid systems," in *Proceedings of the IEEE*, vol. 88, 2000, pp. 971–984.

[5] R. Alur, T. Dang, and F. Ivančić, "Counterexample-guided predicate abstraction of hybrid systems," *Theoretical Computer Science*, vol. 354, no. 2, pp. 250–271, 2006.

[6] E. Clarke, A. Fehnker, Z. Han, B. Krogh, O. Stursberg, and M. Theobald, "Verification of hybrid systems based on counterexmple-guided abstraction refinement," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2003, pp. 192–207.

[7] A. Tiwari and G. Khanna, "Series of abstractions for hybrid automata," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, vol. 2289. Springer, 2002, pp. 465–478.

[8] P. Tabuada, G. J. Pappas, and P. U. Lima, "Composing abstractions of hybrid systems," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, vol. 2289. Springer, 2002, pp. 436–450.

[9] E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald, "Abstraction and counterexample-guided refinement

in model checking of hybrid systems," *International Journal on Foundations of Computer Science*, vol. 14, no. 4, pp. 583–604, 2003.

[10] H. Dierks, S. Kupferschmid, and K. G. Larsen, "Automatic abstraction refinement for timed automata," in *International Conference on Formal Modeling and Analysis of Timed Systems*, 2007, pp. 114–129.

[11] A. Girard, A. A. Julius, and G. J. Pappas, "Approximate simulation relations for hybrid systems," in *IFAC Analysis and Design of Hybrid Systems*, June 2006.

[12] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, vol. 2993. Springer, 2004, pp. 477–492.

[13] S. Sankaranarayanan, H. B. Sipma, and Z. Manna, "Constructing invariants for hybrid systems," *Formal Methods in System Design*, vol. 32, no. 1, pp. 25–55, 2008.

[14] M. Segelken, "Abstraction and counterexample-guided construction of omega-automata for model checking of step-discrete linear hybrid models," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, vol. 4590. Springer, 2007, pp. 433–448.

[15] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HYTECH: A model checker for hybrid systems," *International Journal on Software Tools for Technology Transfer*, vol. 1, pp. 110–122, 1997.

[16] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past hytech," *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 3, pp. 263–279, 2008.

[17] A. Chutinan and B. H. Krogh, "Computational techniques for hybrid system verification," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 64–75, 2003.

[18] E. Asarin, O. Bournez, T. Dang, and O. Maler, "Approximate reachability analysis of piecewise-linear dynamical systems," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, vol. 1790. Springer, 2000, pp. 21–31.

[19] A. A. Kurzhanskiy and P. Varaiya, "Ellipsoidal techniques for reachability analysis," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, vol. 1790. Springer, 2000, pp. 202–214.

[20] A. Girard, "Reachability of uncertain linear systems using zonotopes," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, vol. 3414. Springer, 2005, pp. 291–305.

[21] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, vol. 736. Springer, 1993, pp. 209–229.

[22] C. Moler and C. V. Loan, "Nineteen dubious ways to compute the exponential of a matrix, twenty-five yeasr later," *SIAM Review*, vol. 20, no. 4, pp. 801–836, 1978.

[23] M. Kvasnica, P. Grieder, and M. Baotić, "Multi-Parametric Toolbox (MPT)." [Online]. Available: http://control.ee.ethz.ch/~mpt/

[24] A. S. Tanenbaum, *Modern Operating Systems*, 3rd ed. Prentice Hall, 2007.

[25] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[26] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 2nd ed. Springer, 2004.

[27] Wind River. [Online]. Available: http://www.windriver.com/

[28] QNX Software Systems. [Online]. Available: http://www.qnx.com/

[29] FSM Labs. [Online]. Available: http://www.fsmlabs.com/

[30] C. D. Gill, D. L. Levine, and D. C. Schmidt, "The design and performance of a real-time CORBA scheduling service," *Real-time Systems*, vol. 20, no. 2, pp. 117–154, 1999.

[31] K. Arnold, J. Gosling, and D. Holmes, *Java (TM) Programming Language*, 4th ed. Prentice Hall, 2005.

[32] G. Bollella, B. Brosgol, J. Gosling, P. Dibble, S. Furr, and M. Turnbull, *The Real-Time Specification for Java*, 1st ed. Addison-Wesley, 2000.

[33] Sun Microsystems. [Online]. Available: http://www.sun.com/

[34] R. Henriksson, "Scheduling garbage collection in embedded systems," Ph.D. dissertation, Lund University, 1998.

[35] D. F. Bacon, P. Cheng, and V. Rajan, "The metronome: A simpler approach to garbage collection in real-time systems," in *Workshop on Java Technologies for Real-Time and Embedded Systems*, 2003, pp. 466–478.

[36] M. Iwashiro, K. Furuta, and K. J. Åström, "Energy based control of pendulum," in *Proceedings of the IEEE International Conference on Control Applications*, September 1996, pp. 715–720.